DOCUMENT RESUME

ED 127 970 IR 003 935

AUTHOR	Seidel, Robert J.
TITLE	Project IMPACT: Computer-Administered Instruction Concepts and Initial Development. Technical Report 69-3.
INSTITUTION	Human Resources Research Organization, Alexandria, Va.
SPONS AGENCY	Office of the Chief of Research and Development (Army), Washington, D.C.
REPORT NO	HumRRO-TR-69-3
PUB DATE	Mar 69
CONTRACT	DAHC-19-69-C-0018
NOTE	88p.; For a related document see IR 003 934
EDRS PRICE	MF-$0.83 HC-$4.67 Plus Postage.
DESCRIPTORS	*Autoinstructional Aids; *Computer Assisted Instruction; Computer Oriented Programs; *Computer Programs; Computer Science Education; Evaluation Criteria; Experimental Programs; Information Processing; Instructional Technology; *Military Training; Models; On Line Systems; Post Secondary Education; Time Sharing; Training Techniques
IDENTIFIERS	COBOL; Common Business Oriented Language; Human Resources Research Organization; HumRRO; Instructional Decision Model; Instructional Model Prototypes Attainable Computer; Interactive Computer Systems; Learner Controlled Instruction; Project IMPACT

ABSTRACT

This report summarizes Project IMPACT activities in fiscal year 1968. The goal of the project is to develop a computer-assisted instruction (CAI) training system in the COBOL language for the U.S. Army. Following an introduction, the report explains the instructional decision model which is used with an interactive computer system. The hardware system for Project IMPACT is then explained briefly. COBOL course development is described in terms of training objectives and instructional content. Software developments such as IMPACT CAI LANGUAGE (ICAIL) and future coherent programing are explained in the last chapter. Appendixes include a summary of staff development, a checklist for man-hour computation, a flow diagram from the preliminary COBOL course, COBOL course criterion tests, and an explanation of the IMPACT list processor. (CH)

****
* Documents acquired by ERIC include many informal unpublished *
* materials not available from other sources. ERIC makes every effort *
* to obtain the best copy available. Nevertheless, items of marginal *
* reproducibility are often encountered and this affects the quality *
* of the microfiche and hardcopy reproductions ERIC makes available *
* via the ERIC Document Reproduction Service (EDRS). EDRS is not *
* responsible for the quality of the original document. Reproductions *
* supplied by EDRS are the best that can be made from the original. *
****

ED127970

Project IMPACT: Computer-Administered Instruction Concepts and Initial Development

by

Robert J. Seidel and the IMPACT Staff

March 1969

This document has been approved for public release and sale; its distribution is unlimited.

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Prepared for:
Office, Chief of Research and Development
Department of the Army
Contract DAHC 19-69-C-0018 (DA Proj 2J063101D734)

HumRRO Division No. 1 (System Operations)
Alexandria, Virginia
The George Washington University
HUMAN RESOURCES RESEARCH OFFICE

Technical Report 69-3
IMPACT

TR003935

The Human Resources Research Office is a nongovernmental agency of The George Washington University. HumRRO research for the Department of the Army is conducted under Contract DAHC 19-69-C-0018. HumRRO's mission for the Department of the Army is to conduct research in the fields of training, motivation, and leadership.

The findings in this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

Published
March 1969
by
The George Washington University
HUMAN RESOURCES RESEARCH OFFICE
300 North Washington Street
Alexandria, Virginia 22314

Distributed under the authority of the
Chief of Research and Development
Department of the Army
Washington, D.C. 20310

AD 03832

FOREWORD

This report summarizes the concepts, approach, and developmental activity of the first year's effort (through 30 June 1968) on Work Unit IMPACT, Instructional Model/Prototypes Attainable in Computer Training. IMPACT is an advanced development project undertaken by the Human Resources Research Office, designed to provide the Army with a system for computer-administered instruction. Work on the project is programed for a period of five to seven years.

The research is being conducted in HumRRO Division No. 1 (System Operations). Dr. Robert J. Seidel is the Work Unit Leader and Dr. J. Daniel Lyons is the Director of Research.

The principal contributors to the individual chapters were: for Chapter 1, Mr. George R. Sedberry; Chapter 2, Dr. Robert J. Seidel and Dr. Felix F. Kopstein; Chapter 3, Mr. George R. Sedberry and Dr. Felix F. Kopstein; Chapters 4 and 5, Mrs. Judith G. Compton, Mrs. Beverly R. Hunter, Mrs. Sarah G. See, and Mr. Richard D. Rosenblatt; Chapter 6, Mr. Roy M. Proctor. Mrs. Lola M. Zook provided assistance in organizing and editing the report materials.

The IMPACT project follows on earlier HumRRO work in the same general area under Work Unit METHOD, Research for Programed Instruction in Military Training, and Exploratory Study 42, Organization of Instruction. Principal publications under these research efforts include: The Application of Theoretical Factors in Teaching Problem Solving by Programed Instruction, by Robert J. Seidel and Harold G. Hunter, HumRRO Technical Report 68-4, April 1968 (1); Programed Learning: Prologue to Instruction, by Robert J. Seidel, HumRRO Professional Paper 17-67, April 1967; and Computer-Administered Instruction Versus Traditionally Administered Instruction Economics, by Felix F. Kopstein and Robert J. Seidel, Professional Paper 31-67, June 1967.

Permission has been obtained for the use of copyrighted materials quoted in this report.

HumRRO research for the Department of the Army is conducted under Contract DAHC 19-69-C-0018. The research in Work Unit IMPACT is conducted under Army Project 2J063101D734, Computer-Administered Instruction.

Meredith P. Crawford
Director
Human Resources Research Office

SUMMARY

Problem

Computer-administered instruction (CAI) is a development of vast promise for Army training. It permits broad and precise control of the training environment, and extends the possibilities for *individualized* instruction beyond those offered by any systems for instruction now available. However, a basic principle in training is the fact that the only justification for using any training device or medium, no matter how promising it may be, is its demonstrated capability for enabling a trainee to perform well. Premature or inappropriate attempts to use and evaluate CAI may well hamper or preclude the full development of its potential and the soundly based evaluation of its effects on training. The problem therefore is to devise a system that will provide a controlled training environment and permit appropriate evaluation of CAI for the Army.

Objectives

Project IMPACT is an advanced development effort designed to provide the Army with an effective, efficient, and economical computer-administered instruction system. The objective is to (a) develop two generations of prototype CAI systems with (b) accompanying prototype multipath (branching), individualized programs of instruction.

The system of instruction is to be capable of adapting to the capabilities, at the moment, of each individual trainee. This adaptiveness will be based both on the "entry characteristics" of the trainee and on his long-term and immediate response patterns within the course, so that each step in the instruction will be fitted directly to his needs at that point in the instructional process. The instruction will also be made directly relevant to his specific job requirements.

Courses are to be selected for development on the basis of two criteria: (a) being critical for the Army, and (b) representing widely varied kinds of learning tasks or behavior.

This report summarizes the conceptual and functional activities in the various phases of the research and development plan during the first year of operation for Project IMPACT. The period reported carries through 30 June 1968; information on plans and schedules for the ensuing months is recorded where especially relevant.

Approach

An integrated, interdisciplinary approach is being used in four phased development cycles. Each cycle will produce a number of useful products, designed to provide the Army with its own capability for developing computer-administered instruction. During each of the cycles, developmental activities will be carried on in four interrelated areas: Hardware, Software, Instructional Content, and the model of the Instructional Decision Process.

Work in IMPACT Cycle I was undertaken at the beginning of FY 1968. The research and development effort for the total project was originally programed for a five-year period, but attenuation of funding has resulted in tentative reprograming over six and one-half years. Cycle I completion is now forecast for the end of FY 1970.

During this initial stage, the IMPACT work is dealing with construction of a CAI course in the COBOL computer programing language. This course was selected for prototype development because of indications from many Army sources of the great need for informed use of computers and greater sophistication in data processing. HumRRO already had subject matter expertise through earlier development of a programed instruction course on computer programing fundamentals in Work Unit METHOD.

IMPACT Cycle I efforts are directed toward integrating many different kinds of developmental activities to produce a provisional, first-generation CAI system. In IMPACT Cycle II, the "breadboard" model will be tested for training effectiveness and revised into an operationally implementable CAI system. In IMPACT Cycle III, a second-generation CAI system will be designed, incorporating improvements in hardware and software, and especially in the Instructional Decision Model (IDM), and developing additional prototype courses with a wider variety of training content and learning tasks. During IMPACT Cycle IV, the effectiveness of the second-generation CAI system will be tested, and a third-generation system with expanded and upgraded capability will be designed.

Progress During FY 1968

A professional and technical staff from the fields of behavioral science, applied mathematics, computer science, and instructional programing was formed. Their developmental activities on the various facets of a CAI system during FY 1968 may be summarized as follows:

Instructional Decision Model. This model, the heart of a computer-administered instruction system, will be a set of rules for matching presentation of specific content (selecting and sequencing) with trainee capabilities (student characteristics and responses to earlier material). During this year, "breadboard" development of the model began to take shape. Various aptitude, achievement, and other diagnostic tests were surveyed and a suitable set selected. Concepts, methods, and techniques from the psychological literature were selected for adaptation to IMPACT requirements. The analysis of the subject-matter structure of COBOL was begun according to rigorous formal (mathematical and logical) techniques. These and other source materials allowed work to begin on the preliminary formulation of decision rules for use in individualizing instruction. (See Chapter 2.)

Hardware.[1] Hardware being used in Cycle I is basically an IBM model 360/40 computer and, at the student terminals, Sanders 720 cathode ray tubes with light pens. A secondary visual display is being supplied at the terminals by means of the Perceptoscope, a type of 16mm projector. One sensitized electronic tablet with an electronic pen has been procured for experimental use at one terminal; this tablet provides a natural handwriting (block printing) input device. Initial developmental work in the early stages was carried on with typewriter terminals. (See Chapter 3.)

Instructional Content. Detailed behavioral objectives for the COBOL instructional course were developed, and the portion of the instructional content to be used in the first student testing of the prototype development was completed. An analysis of the Army jobs for military and civilian COBOL programers provided the basis for the development of the behavioral objectives, and the instructional content was then prepared on the basis of these objectives. Criterion tests related to the objectives were developed. (See Chapters 4 and 5.)

Software. Software development was designed with the view of satisfying both the initial requirements of the project and, with modifications, the requirements of the later developmental cycles. A computer-administered instruction language, IBM Coursewriter, was modified to run IBM 1050 typewriter terminals for the IBM 360/40 computer disk operating system. Development of a limited list-processing capability was begun within Coursewriter as a major project effort. Modification of Coursewriter for use with the cathode ray tube was virtually completed. (See Chapter 6.)

---

[1] Identification of products is for research documentation purposes only; this listing does not constitute an official endorsement by either HumRRO or the Department of the Army.

CONTENTS

Chapter Page

1 **Introduction** 3
- Objectives 3
- Background and Rationale 3
- Plan of Development 4
- Summary of Progress During FY 1968 6

2 **Preliminary Conceptualization of the Instructional Decision Model (IDM)** 8
- Approach and Rationale 8
- Some Issues in Designing an IDM 10
  - A General Overview of Decision Models 10
  - The Student Image 11
  - The Subject-Matter Map 11
- Issues Addressed in First Implementation 12
  - Subject-Matter Structure 12
  - Student Knowledge States 14
  - Empirical Study of First IDM 17
- Iterative Elaboration of IDM 18

3 **Description of the Hardware Base of the IMPACT System** 22
- Hardware Subsystems 22
  - Information Processing Subsystem 22
  - Data Storage Subsystem 22
  - Communications Subsystem 22
- Developmental and Experimental Stations 24

4 **COBOL Course Development I: Objectives** 26
- Introduction 26
- Establishment of Training Objectives 26
  - Tasks 27
  - Terminal Objectives 28
- Characteristics of the Job 29
  - COBOL in the Army 29
  - Job Specifications of COBOL Programers 30
- Standards and Evaluation 31
- Characteristics of the Programer Trainee 31

5 **COBOL Course Development II: Instructional Content** 36
- Introduction 36
- Outline of Content and Organization 37
- Tests of Proficiency 38
  - Criterion Tests 38
  - On-Line Multiple-Choice, Constructed-Response Subtests 38

Chapter Page

Response Analysis and Data Collection in Preliminary COBOL Course 39
Definitions 39
General Design Considerations 40
Specific Plan: Response Analysis 40
Specific Plan: Data Collection 41
Preliminary Data Collection and Expansion of Content 42
Subject Selection for Preliminary Data Collection 42
Entry Characteristics Testing Program 43
Individualization of Instruction 43

6 **Software Developments and Projections** 44
Introduction 44
Major Software Development Efforts 44
Technical Introduction 45
ICAIL—CRT Conversion 47
Functions 50
Instructional Decision Model Data Structures 53
Future Systems and CAI Software Development Plans 57
Coherent Programing 58
Other Factors 59

Literature Cited 63

Appendices

A Summary of Staff Developments by Quarter 65
B Supervisor's Checklist for Manhour Computation 67
C Flow Diagram From Preliminary COBOL Course 69
D COBOL Course Criterion Tests 76
E The IMPACT List Processor 78

Figures

1 Planned Schedule of Development 5
2 IMPACT—Development Activities During Cycle I 5
3 The Instructional Situation 9
4 Hypothetical Knowledge Space 13
5 Data Structure 15
6 Valid Confidence Scoring of Constructed Responses 19
7 HumRRO Hardware Configuration 23
8 General Overview of ICAIL—List Processor Translator 49
9 Similarities and Differences Among Four Functions 52
10 The Projected IMPACT Software System 58

Chapter 1

INTRODUCTION

OBJECTIVES

Project IMPACT is an advanced development effort designed to provide the Army with an effective, efficient, and economical computer-administered instructional system (CAIS). The system being developed will include prototype individualized (branching) programs of instruction. Ultimately, the project will provide the Army with its own capability for developing computer-administered instruction; the system will be designed and documented for use by technically unsophisticated personnel, such as instructors or subject matter experts, so that they will be able to modify or develop materials for their own purposes and for new courses.

This system for instruction will be able to adapt to the capabilities, at the moment, of each individual trainee. This adaptiveness will be based on both the entry characteristics of the trainee and his long-term as well as immediate response patterns within the course; each step of instruction will, thus, be fitted to his needs at that point in the process. The instruction will also be made directly relevant to his specific job requirements.

During the first stage of the development effort, the research has dealt with construction of a CAI course in the COBOL computer programing language. Additional prototype courses selected for later development will meet the same criteria by which the original subject matter was selected: (a) being critical for the Army, and (b) representing unique kinds of behavior.

This report summarizes the conceptual and functional activities in the various phases of the development plan during the first year of operation for Project IMPACT. The period reported carries through 30 June 1968; information on plans and schedules for the ensuing months is recorded where especially relevant.

BACKGROUND AND RATIONALE

One of the fundamental principles of training is that the only justification for use of any training device or medium, no matter how promising it may appear to be, is a demonstrated capability for enabling a trainee to perform well. Computer-administered instruction is a development of undeniable and vast promise. However, premature or inappropriate attempts at using or evaluating CAI can only arrest or preclude the full development of this potential. No matter what the capacities of the information processing machines, they do not guarantee relevant decisions that will lead to effective instruction.

Project IMPACT is therefore pursuing an evolutionary approach toward developing an effective and operationally implementable CAI system. The key factor in such a system is its potential for extending the possibilities for individualized instruction beyond those offered by available methods of instruction. The computer provides an immense capability for information acceptance, for rapid storage and retrieval of vast quantities of data, and for rapid, continuing, coordinated execution of multi-factored decisions. It thus becomes possible to adapt instruction far more precisely to the needs of each individual trainee than can be done with existing systems for instruction.

For example, efficient instruction must seek to teach a trainee at any given moment that which he has not yet learned and that which is a next step in his training. Programed instruction can adapt the instructional presentation to the gross characteristics of a specific individual. CAI, however, can potentially adapt the instructional presentation not only to much more detailed characteristics of the trainee but also to his precise requirements of the moment—provided that the decisions that govern the selection, sequence, and manner of presentation of instructional "units" are soundly based on the fundamental principles of human learning. It must always be remembered that it is the instructional process that determines the efficiency of training; computer hardware and software are merely implements for setting the process into motion.

The essence of CAI is therefore the instructional decision model (IDM)—a set of rules for matching presentation of specific content (selecting and sequencing) with trainee capabilities (student characteristics and responses to earlier material). However, since the computer is the instrument for gaining broad and precise control of the training environment, the capabilities and limitations of the computer's hardware and software do provide constraints on the decision model.

Selection of portions of a course in computer programing for the initial CAI developmental work was based on indications from many Army sources of the great need for informed use of computers and greater sophistication in data processing. The computer language chosen for the course was COBOL because it appeared to be the language with the greatest range of applicability to general Army problems, and has the flexibility for use both in small- and large-core computers.

PLAN OF DEVELOPMENT

An integrated, multi-faceted, and interdisciplinary approach is being used in the CAI developmental plan, which involves four phased development cycles. The developmental activities during each of the cycles are grouped under four major classifications: Hardware, Software, Instructional Content, and the Instructional Decision Model.

Work in Cycle I was undertaken at the beginning of FY 1968. The development effort for the total project was originally programed for a five-year period, but attenuation of funding has resulted in a tentative reprograming over six and one-half years. Under this schedule, completion of Cycle I is forecast for the end of FY 1970. The long-term schedule as presently envisaged is shown in Figure 1.

Work in Cycle I is essentially devoted to the assembly of a "breadboard" version of the initial CAI system. Developmental activities on the Instructional Decision Model, the Instructional Content, the Hardware, and the Software components of the system are under way, and the initial versions of the various products are being provisionally integrated and adjusted to each other. These activities have been facilitated by being able to draw upon subject matter expertise gained by HumRRO in previous research (Work Unit METHOD) which dealt with development of a linear programed course in fundamentals of computer programing. The nature and interrelationships of the various activities in Cycle I are illustrated in Figure 2; the expected products of the cycle are also listed there.

Planned Schedule of Development

	FY 68				FY 69				FY 70				FY 71				FY 72				FY 73				FY74	
Cycle I	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2
Hardware				▨	▨	▨	▨	▨																		
Software	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨														
Instructional Content	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨														
Decision Model	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨																
Cycle II																										
Hardware												▨	▨	▨	▨											
Software										▨	▨	▨	▨	▨	▨	▨	▨	▨								
Instructional Content												▨	▨	▨	▨	▨	▨									
Decision Model													▨	▨	▨	▨	▨	▨								
Cycle III																										
Hardware														▨	▨	▨	▨	▨	▨							
Software																▨	▨	▨	▨	▨						
Instructional Content															▨	▨	▨	▨	▨	▨	▨	▨				
Decision Model															▨	▨	▨	▨	▨	▨	▨					
Cycle IV																										
Hardware												▨	*						▨	▨	▨	▨	▨	▨		
Software																	▨	▨	▨	▨	▨	▨	▨	▨	▨	
Instructional Content																		▨	▨	▨	▨	▨	▨	▨	▨	
Decision Model																			▨	▨	▨	▨	▨	▨	▨	▨

*Preparation of specifications for purchase of second generation hardware.

Figure 1

IMPACT — Development Activities During Cycle I

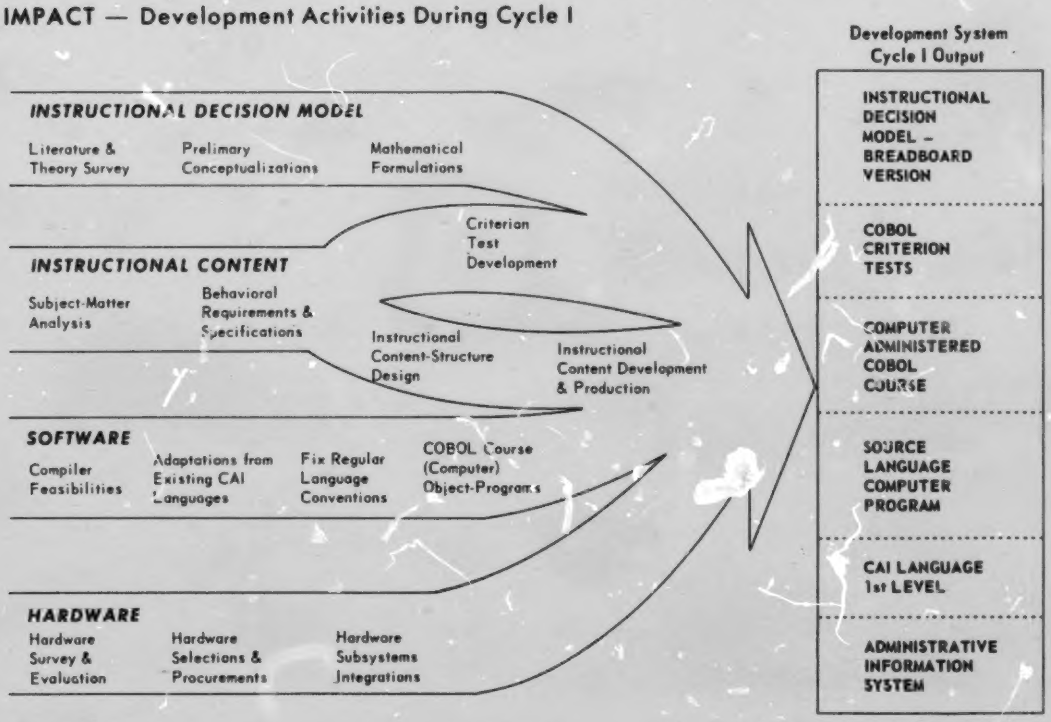

Development System
Cycle I Output
INSTRUCTIONAL DECISION MODEL
Literature & Theory Survey
Prelimary Conceptualizations
Mathematical Formulations
Criterion Test Development
INSTRUCTIONAL CONTENT
Subject-Matter Analysis
Behavioral Requirements & Specifications
Instructional Content-Structure Design
Instructional Content Development & Production
SOFTWARE
Compiler Feasibilities
Adaptations from Existing CAI Languages
Fix Regular Language Conventions
COBOL Course (Computer) Object-Programs
HARDWARE
Hardware Survey & Evaluation
Hardware Selections & Procurements
Hardware Subsystems Integrations
INSTRUCTIONAL DECISION MODEL – BREADBOARD VERSION
COBOL CRITERION TESTS
COMPUTER ADMINISTERED COBOL COURSE
SOURCE LANGUAGE COMPUTER PROGRAM
CAI LANGUAGE 1st LEVEL
ADMINISTRATIVE INFORMATION SYSTEM

Figure 2

Cycle II involves a test of the "breadboard" version of the CAI system developed in Cycle I, to determine its effectiveness in training. On the basis of evaluation experience, this version will then be revised into the first-generation prototype of an operationally implementable CAI system.

Cycle III and IV activities will depend to a considerable extent on the experience gained in Cycles I and II. In general, during Cycle III, in coordination with operational tests and evaluation of installations of the initial prototype course, designs for a second-generation CAI system will be developed. This design will incorporate improvements in hardware, in software, and especially in the model of the instructional decision process (to include improved indices of decision factors, additional decision factors, and an expansion in range and depth of the operating decision rules). In choosing instructional content for the development of additional prototype courses, emphasis will be placed on choosing instruction reflecting a wide variety of learning tasks (e.g., instruction in a psychomotor skill). Operational implementation of a provisional second-generation CAI system may be feasible.

During Cycle IV, there are plans for tests of the effectiveness of the second-generation CAI system, and designs for a third-generation system with upgraded and expanded capability. Effectiveness tests will begin to assess long-range effects of CAI use, to provide feed-in for improvements in the third-generation system.

SUMMARY OF PROGRESS DURING FY 1968

In the first year of work on Project IMPACT, a major activity was that of recruiting an interdisciplinary staff to meet the varied requirements of the project. Gradually, as the various elements of the work were activated, a staff was assembled representing the fields of behavioral science, computer science, applied mathematics, and instructional programing. A summary of staff developments through 30 June 1968, by quarter, is shown in Appendix A.

Another major activity during the first year was that of developing the specialized physical facilities required for various aspects of the IMPACT work. In addition to providing appropriate facilities for the professional and the technical support staffs, design and construction were begun for 12 student stations for use in the later developmental phases of Cycle I. The cubicle construction was scheduled for completion in the first quarter of FY 1969.

Activities in developing the four major components of the CAI system, during the first year of Project IMPACT's existence (through 30 June 1968), are summarized in the remainder of this report:

Chapter 2 deals with the progress that has been made in conceptualizing the Instructional Decision Model, which is the heart of the computer-administered instructional system.

Chapter 3 is a brief description of the computer and related hardware that has been selected for the first-generation IMPACT work. A chart of the hardware configuration is included. This chapter provides a framework for the discussions of instructional content and software in the remaining chapters.

Chapters 4 and 5 deal with the instructional content component of the system. Development of objectives for the COBOL course is described in Chapter 4, and the preliminary work in the development of course content is described in Chapter 5. Appendices provide various types of illustrative or documentary material for these activities.

Chapter 6 describes the activities that have been directed toward meeting the software needs for the current developmental work on the Instructional Decision Model and the COBOL course. At the same time the software staff has begun to project needs and develop programs for future activities, so that these facilities will be available as they are needed at progressive stages of model and course development.

Chapter 2

PRELIMINARY CONCEPTUALIZATION OF THE INSTRUCTIONAL DECISION MODEL (IDM)

This chapter deals with the development that is the key to the work in Project IMPACT, and to the products that may be expected from it—the construction of a model to guide the making of decisions on presentation of instruction. The chapter describes the rationale of the IMPACT approach to conceptualizing the instructional process in a form that will be implementable via a computer. It summarizes some major issues raised by the chosen approach, and describes the progress that has been made in solving the major problems in terms of both current and projected implementation. Finally, both immediate and long-term prospective improvements in the IDM are described.

APPROACH AND RATIONALE

The essential premises underlying the IMPACT approach to the development of computer-administered (or -implemented) instructional systems are summarized below. The rationale is discussed further in Seidel and Kopstein (2).

Traditional psychological learning theory includes the notion that a learned connection between stimulus and response is a simple, stable, functional relationship, and that the purpose of instruction is to establish such relationships in order to guarantee that presentation of a stimulus uniformly leads to the execution of the desired response.

The instructional situation is viewed from a different perspective in the IMPACT work. The concept is that of an open information-exchange system, in which there is continual interaction between the student and his environment, and in which the effective environment for the student changes as his characteristics change with continued experiences.

This view of the instructional situation provides IMPACT with its general framework for the development of the instructional model. It takes into account (a) the information made available to the student, (b) the information he assimilates, and (c) the information he actually puts to use within the constraints of the problem-context imposed on him. He receives information (including error feedback) from the environment and he transmits information (including error output) to the environment. The information-exchange instructional model encompasses the adaptive requirements—that is, change as a consequence of interaction—pertinent to efficient development of performance capabilities by the student.

Carrying this thought a step further, with reference to developing a model of the instructional process: Research on human learning, including programed instruction, has generally dealt with small-scale, artificial laboratory environments where the student learns to repeat a list of simple materials (e.g., lists of nonsense syllables); many studies have compared various conditions for learning such material. Behavioral science has provided a store of experimentally established generalizations (i.e., regularities) relevant to instruction, but most were established in laboratory settings, not for an interactive, adaptive environment. The store of knowledge in such settings does not deal with instructional information as it is usually encountered and used by the student—a highly complex, structured set of interrelated concepts, commonly in paragraph-like format.

The information interchange approach is schematized in Figure 3. An instructor (instructional agent) sends instructive messages over the "teach channel"; his objective is to engender and control a gradual change process in the student that will lead to a final proficient state. The student, in turn, sends messages to the instructor over the "test channel"; these communications always report his state or his progress at that moment toward specified instructional goals. The dotted line cutting through both channels serves to identify the regions in which the communicative process is under direct control by the instructor or under direct control by the student.

The Instructional Situation

(Schematic Representation)

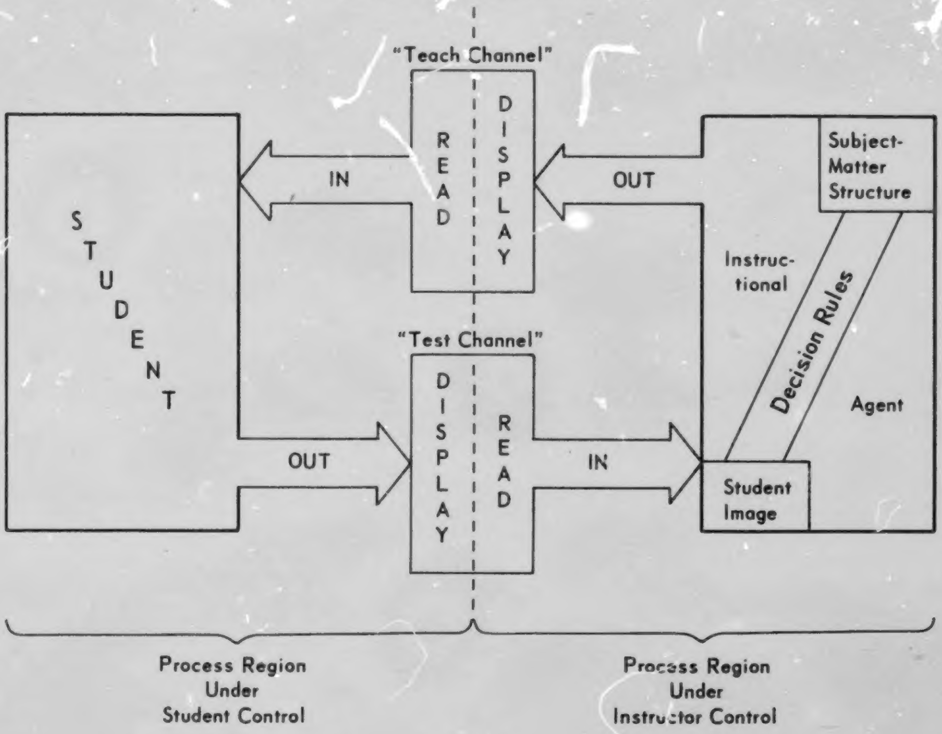

"Teach Channel"
READ
DISPLAY
IN
OUT
STUDENT
Subject-Matter Structure
Instructional
Decision Rules
Agent
"Test Channel"
DISPLAY
READ
OUT
IN
Student Image
Process Region Under Student Control
Process Region Under Instructor Control

Figure 3

The instructor's direct control over the "teach channel" is limited to displaying information. The student may read, transform, accept, or reject none, some, or all of this information; the instructor cannot absolutely determine which will be the case. Similarly, for the "test channel" the student can display as much of the information about his momentary state as he wishes, but he cannot assure that all of this information will be assimilated by the instructor.

The object of instruction is maximum end-of-course proficiency. At the end, one might say, the student can do anything the instructor asks and the instructor knows this to be true. For effective and efficient instruction to take place, the information flow within each instructional step must be optimized—that is, the "best" strategy for transmitting information must be followed.

This requirement for an optimal information-flow strategy provides the specifications for an instructional decision model (IDM). An IDM is simply a set of rules for action over time under a range of circumstances—in other words, an IDM provides the basis for repeated decisions to select appropriate courses of action. For purposes of evaluation and progressive improvement, the rules that guide action choices must be *minutely explicit*.

SOME ISSUES IN DESIGNING AN IDM

This summary report must be confined to presenting only a few of the major (at present) issues in devising an optimal strategy for transmitting information—or an effective and efficient IDM. In this section, some general characteristics of decision models, the nature of the image of a student, and the importance of establishing a structure of subject matter will be discussed. In the following section, first attempts at the solution of these issues will be described.

A General Overview of Decision Models

For purposes of discussion, a decision model may be viewed in terms of three aspects:

(1) Decision factors
(2) Decision options
(3) Decision rules

Decision factors are the *conditionals*. They are the things that would follow the "if" in a statement such as: "If *a* (or *b* or *c* or *d*) is the case, then *x* (or *y* or *z*) . . .". They are the *a*, *b*, *c*, *d* to be considered. In the context of instruction, we may loosely group the decision factors as:

(1) The kind of subject matter to be taught.
(2) The kind of student to be taught.
(3) The circumstances under which the teaching is to take place.

Decision options are the *available actions*. In the "If . . . , then . . . " sentence, they can be equated with *x*, *y*, *z*. These options refer only to courses of action under the *direct* control of the acting agent; considering courses of action over which only indirect control may be exercised is largely futile. For instruction, the decision options are choices such as:

(1) The substantive information to be presented.
(2) The amount of such information.
(3) The form in which it will be presented.
(4) Auxiliary (e.g., attention directing) information to be presented.
(5) The form of the auxiliary information.
(6) The channel(s) of information presentation (e.g., auditory, visual).

Decision rules are the *link* between decision factors and decision options. They are the way(s) the decision factors are to be separated or linked to determine the grouping and connections among decision options; for example, "If *a* and *b* but not *c* or *d* . . . , then *x* or *y* (but not both) and *z*." If the instructional process under examination changes over time, then a sequential decision-making model is needed. It may be necessary to assign differential importance to certain decision factors and options at different times. Through iterative, developmental testing, the possible ways of grouping will be reduced to a formal set of rules.

The Student Image

While the above outline suggests how an IDM can be used to reach and execute instructional decisions, it does not show how the IDM ascertains what "kind of student" is being taught. The answer is a model within a model; it will be termed "the student image."

Returning to Figure 3, let us assume that the instructor will be not a human being but a computer program. This program (the instructional agent) initially has no direct knowledge about the student (ignoring, for the moment, the fact that information could be prestored). Its only available means for determining the student's characteristics is to observe the relationships between information it (the computer program instructional agent) displays and the information it receives in response from the student.[1] The fact that the instructional agent's knowledge of the student must be indirect makes it necessary for the instructional agent to form a representation—an image—of the student, in order to be capable of interacting with him in a planned and effective fashion.

The image for a particular student is formed over time and gradually becomes more and more precise and reliable. It is possible to prestore information about a student's entry characteristics—his aptitudes, his prior achievements, his personal attributes. However, such a prestored image cannot contain details about current characteristics because such information does not exist until a sequence of instructional interactions has begun.

The instructional agent needs information as to how the student characteristically copes with information being presented, and what information he requires in order to cope successfully. As will be seen later, the work of Guilford (4) suggests the way to describe how the student copes. The work of Harary, Norman, and Cartwright (5), as well as that of Gagné (6), suggests the way to describe what information is required.

The Subject-Matter Map

A parallel to the problem of the non-specific student exists for the subject matter to be taught. Simply storing an encyclopedic stock of information about the subject does not solve the problem. The material must be organized in a way which will provide the decision model with a basis for deciding which facts, principles, or procedures, are built on which other facts, principles, or procedures, what interrelationships exist among them, and which topics may be best presented before or after which other ones. There must be a consistent "map" to represent the subject matter and its organization, and it must be more abstract than the subject matter itself. Without a map, the IDM cannot tell where it is starting, where it is going, where it has already been, and how it will get to where it wants to go.

Actually, there is a dual requirement: first, for an abstracted representation of the subject-matter organization—the subject-matter map; second, for a set of clear, consistent, and concise rules to provide each specific student with his individual route for traversing this map.

---

[1] We see this perhaps even more clearly if we apply the test proposed by A.M. Turing (3), in which the human student is replaced by a computer program capable of mimicking his behavior. In these circumstances the program that constitutes the instructional agent could not distinguish between the "real" student and the "artificial" student.

The term "map" is used to refer to a model or independent image of subject-matter organization (structure), because it is applicable in the ordinary sense of the word as well as being technically correct. The subject-matter maps (see the following section) will be mathematical graphs and nets, which consist of points (also called nodes) and of lines (also called arcs) connecting the points in various ways. In a mathematical sense, roadmaps that show roads connecting points representing cities and towns are actually graphs. In the roadmap, it is very clear what the points and lines represent; this is not so clear in the case of subject-matter structure. Clarifying this structure and translating it into a proper model is one of the most challenging problems in applying topological mapping to instruction.

ISSUES ADDRESSED IN FIRST IMPLEMENTATION

From establishing a rationale for the basic design approach in Project IMPACT, we now turn to describing preliminary approaches and progress made. The information that follows is a record of progress in conceptualization or developmental application. With further experience, some of these approaches will be found faulty or inadequate; they are most usefully viewed as points of departure or "best estimates."

First, the representing or "mapping" of subject-matter structure will be discussed. Second, the first attempts at forming the student image via a more sensitive probe of the student's knowledge states will be described. Next, implementation of the inquiry technique, and finally, the initial decision strategies to be used will be described.

Subject-Matter Structure

Two broad premises provided the starting point for the effort to develop a way of representing and organizing the structure of subject matter to be taught. First, any given subject matter is an arbitrarily delimited portion of human knowledge. Second, just as knowledge in general is incomplete and is constantly increasing, so is knowledge of a specific subject matter. Increase in knowledge takes place when new facts are discovered or new concepts are established, or when new relationships among facts and/or concepts are discovered.

Thus, we need to distinguish among several levels of knowledge: (a) the potential knowledge existing in the universe, (b) the portion of that knowledge that is possessed by mankind so far, (c) an arbitrarily delimited subject-matter area within the totality of existing human knowledge, and (d) the knowledge about a given subject-matter area possessed by a specific individual or group (the instructor or instructional staff). The information about a subject matter, such as COBOL, possessed by even the most knowledgeable instructor is a finite and even quite limited structure.

Further, instruction is concerned only with communicable knowledge. Private knowledge that cannot be communicated or independently verified is of no use in designing an instructional decision model. For knowledge to be communicable, it must first be stored by the instructor or instructional agent in memory (human or computer), and it must be selectively retrievable. The "map" or index of the subject-matter structure is what makes it possible for selective retrieval to take place.

Instruction can be defined as transferring the instructor's (instructional agent's) knowledge structure to the student. For a delimited knowledge structure (the "arbitrary" requirements for a course), instruction has been a success when one cannot distinguish (in a loose sense) between the instructor and the student. That is, ideally the student should have learned all that the instructor knows about the subject matter in the course, and should be equally able to answer questions or perform tasks pertinent to the course requirements.

Even an approximation of this ideal state is not readily attained. The knowledge structure of the instructor is communicated to the student over a period of time. In the early stages of instruction, only a small portion of the instructor's knowledge structure will be transferred to the knowledge space of the student; it will be limited in extent (i.e., awareness of interrelations). Efficient instruction transfers those facts and concepts that have not yet been effectively stored by the student, and establishes those interrelationships that produce maximal strength in minimal time. The instructional strategy to accomplish this transfer—that is, the instructional decisions made sequentially over time—is determined by the nature of the subject matter, by the characteristics of the student (viewed as a system for accepting, storing, and retrieving information), and by the specific history of the instructor-student interaction.

The distinctions to be made among the several versions of a subject-matter structure can be viewed as in Figure 4. The circles (nodes) represent the concepts of the knowledge space, and the lines (arcs) represent the relations among the concepts. The "Current Ideal" label reflects the fact that knowledge of any given subject matter is incomplete and constantly "growing." As for the "Instructor Understanding": This instructor does not have all the knowledge about the subject matter that can be possessed currently, but he has enough to serve the "Course Requirements"; he also has more relational awareness than is needed to convey the course properly to the students. Finally, "Innovative Student" has successfully completed the course requirements; in addition, he has independently discovered a new relation, not known to his instructor, and not even existing within the "Current Ideal" structure of the subject matter.

This example illustrates the necessity for taking into account (a) relative states of knowledge about a given subject matter (e.g., COBOL) for instructor and student, and (b) potential individual differences among students. Implicit in it is the notion of generalization—a transfer capability that will engender proficiency in tasks not yet encountered. This capability is important because the course requirements, as depicted, provide only a subset of the relations that could be required if a more complete mastery of the subject matter were desired.

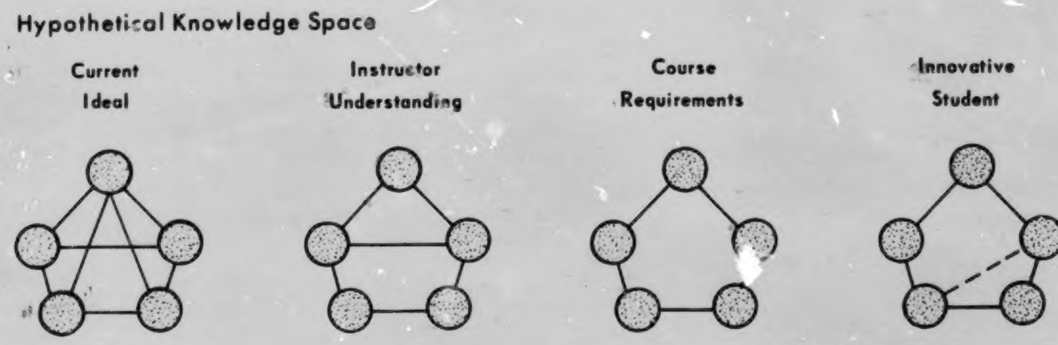


Figure 4

The next question is precisely how to express the notion of conceptual points (nodes) and relationships (arcs) among these concepts, in order to specify subject-matter structure or organization for an instructional decision model. If a subject-matter structure can be expressed precisely, it can be expressed mathematically. If it is expressible mathematically, there is an adequate form of expression for the computer; that means the subject-matter structure can be implemented within the instructional decision model and within the computer.

The type of mathematics that seems most useful for this purpose is topology—more specifically, the mathematics of nets and graphs, of which maps are a special case. Representing subject matter in terms of nets and/or graphs has advantages. First, nets and graphs consist of points (nodes) and lines (arcs); points can be used to represent concepts, and lines used for relations. Second, nets and graphs can be converted directly into set theoretic notation, and into matrix algebra.[1]

For IMPACT's IDM, the problem is how to interpret graphs in terms of subject-matter structure or to impose graph and net formal structures onto empirical reality. Looking at it in another way, it is the problem of establishing valid coordinating definitions between nodes and arcs in a graph and the empirical (or subject-matter) structures they are to represent.

Communicable knowledge (e.g., the structure of COBOL) is itself an abstraction of an underlying empirical reality. Acceptance of this basic proposition implies that, for IDM use, factual statements about a subject-matter (e.g., COBOL) must be capable of being framed according to the principles of the predicate calculus, a branch of mathematical logic. It is evident that in order to be able to do this, we need more rigorous and more powerful analytic procedures than those which have hitherto been employed in the analysis of subject matter (e.g., Taber, Glaser, and Schaefer (9).

This search for the needed procedures is taking the form of an effort to provide a basis for ordering the relations among the concepts of the COBOL subject matter being explored in Cycle I of Project IMPACT. Digraph and net theory is being applied to hierarchically ordered sets of terms and relations in that subject matter. In addition, a way of measuring the distance between concepts is also being explored.[2] In other words, the objective is to specify the mathematical basis for establishing a potentially meaningful, instructional distance between any two sets whose elements are interpreted as subject-matter terms (e.g., concepts of "file name" and "data division" in COBOL). Implications of this approach for studying linguistic structures in software for CAI are also being considered.

Student Knowledge States

The heart of computer-administered instruction is an Instructional Decision Model that makes it possible to determine which information should be presented how and when to a particular student. CAI, with its potential for total control of the instructional environment, is viewed as a logical extension of programed instruction, but it can adapt itself much more precisely to the student—especially with respect to his informational requirements of the moment.

---

[1] For an introduction to the mathematics of graphs, and particularly directed graphs or digraphs, the reader is referred to Harary, Norman, and Cartwright (5), Flament (7), and Berge (8).

[2] The approach is based on the set theoretic principle that the symmetric difference between two finite sets with denumerable elements meets the axiomatic requirements for a metric (10).

Central to this premise is the degree to which the adaptive IDM accurately describes the student's knowledge states over time. For the present IDM(s), the data elements and their structuring for analysis will be as indicated in Figure 5. The data structure will be studied within the following comprehensive framework for examining the student's (a) dimensions of information processing, (b) dynamic organization of the subject matter, and (c) confidence in his responses.

Data Structure

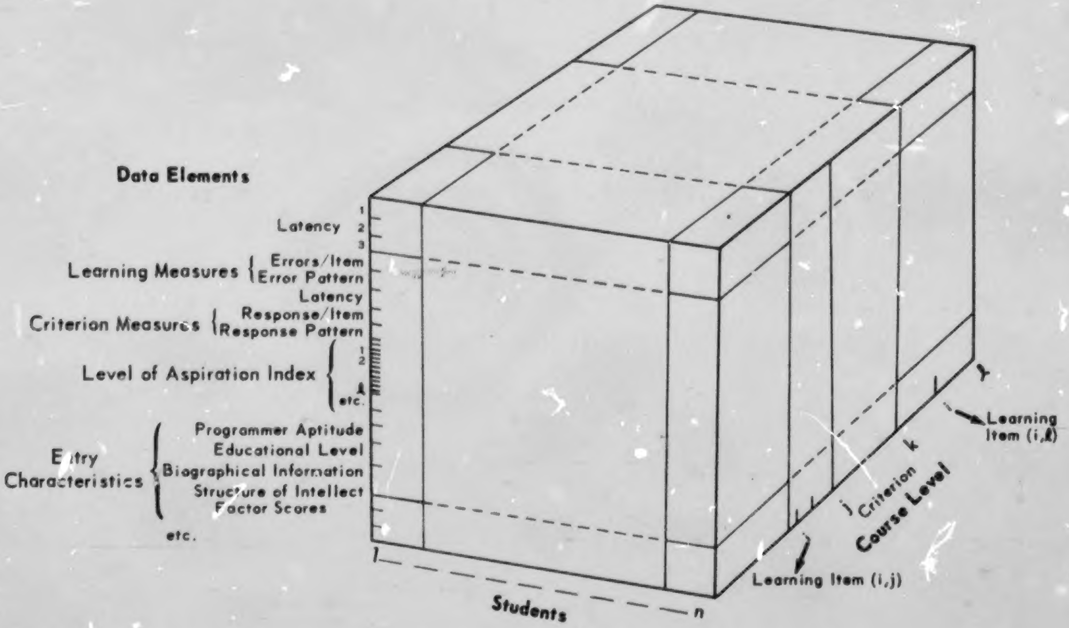

Data Elements
Latency
1
2
3
Learning Measures
Errors/Item
Error Pattern
Latency
Criterion Measures
Response/Item
Response Pattern
Level of Aspiration Index
1
2
etc.
Entry Characteristics
Programmer Aptitude
Educational Level
Biographical Information
Structure of Intellect Factor Scores
etc.
Learning Item (i,ℓ)
k
Criterion
Course Level
j
Learning Item (i,j)
1
Students
n

Figure 5

The kinds of data to be gathered and the framework within which they will be analyzed are indicated in Figure 5.[1] The vertical dimension represents the specific characteristics of the individual student—the measures mentioned earlier to describe each person with respect to his abilities (including structure of intellect factors described hereafter), achievements, response latencies, error patterns and rates, and so forth. Where feasible within the constraints of the software/hardware subsystems, the response latencies will be broken down into three components: reading time (Latency 1), start-response (Latency 2), and stop-response (Latency 3). The horizontal dimension in Figure 5 represents all of the individuals included within the student group. The remaining dimension represents segments or slices (such as item i,j) through the course of instruction in serial order.

It will be seen that this data structure makes it possible to relate any one, or all, of the measures representing a given student, either to the comparable measures for all of the students, or to the comparable measures within prior course segments, or both.

---

[1] The data management software systems being worked out for analysis of these data are described in Chapter 6 of this report.

Dimensions of Information Processing by the Individual. The framework for this approach is the recent theoretical and experimental work of J.P. Guilford and associates (4, 11). Guilford has developed a model of human intellectual functioning based upon an ordered array of abilities, which are acquired (learned) and relate to the processing of information. The model is three-dimensional—Content, Operations, and Products. In essence these dimensions correspond, respectively, to form of information input, types of operations to be performed upon this input, and the type of output (end-state, product) to be produced as a result of the processing.

Subdividing the dimensions according to the array of abilities results in 120 cells, each representing a specific ability component in human intellectual functioning. Any individual will possess each of these abilities to a particular degree. Since the Guilford approach characterizes the particular types of process by which the student handles or transforms particular types of transmitted instructional information, it is an excellent complement to the structural approach to subject matter described earlier.

Empirical support for Guilford's model for the structure of intellect comes, first of all, directly from his and his co-workers' data substantiating the existence of over two-thirds of the abilities predicted by the model. In addition, past experimental work on instruction (Seidel & Hunter, 1) has shown that aptitude tests administered prior to instruction predict performance progressively less well in successive course segments. However, predicting performance in the next course segment on the basis of achievement test scores on the current segment progressively improves.

Observations such as the latter led Bunderson (12) to hypothesize that different abilities are active at different stages of learning (or at different stages of progress in mastering the task). Bunderson as well as Dunham et al. (11) have produced considerable evidence in support of this hypothesis.

The above analysis implies that an instructional decision model should match the particular instructional presentation—at any given stage of progress toward mastery—to the individual's configuration of abilities with respect to the kind of information processing (intellectual functioning) involved at that point. There is, thus, a requirement for a model of the subject matter from which to determine the kind of intellectual functioning that will take place, when, and where.

Dynamic Organization of the Knowledge Space. At any moment, the IDM may be required to make a decision on how to present the next instructional "slice." For every action of the student, there is a "best" reaction (or class of reactions) that the instructional agent can make. The problem of making these decisions of the moment, consistently and in an optimal manner, makes it clear that the student needs to have a good capability for making inquiry. To the extent that "action" by the student is restricted—that is, he cannot inform the system adequately of his state and his needs of the moment—decisions by the instructional agent will be less than optimal.

As a first step for this facet of the IDM, a glossary technique (termed INFO) has been developed. As the student is exposed to each conceptual unit, he may ask about unfamiliar terms. He will then be given information about the terms as requested, and the IDM will be given information concerning the student's structure or organization (or the lack of it) of the material as it is presented.

Response Confidence. In conventional testing, the student is usually trying to make a high score rather than to indicate his knowledge about the subject matter. His interests (graduation, honors, awards) are not best served by revealing a lack of knowledge; thus, under such circumstances potential remedial action is not pursued. Traditional testing thus is often viewed as a purely competitive interaction between student and instructor. Moreover, scores on traditional types of tests primarily relate the student's performance to that of other students (norms) and give little information about the degree of the student's subject-matter mastery. In contrast, the IMPACT IDM uses all measures as sequential, diagnostic estimates of the student's knowledge state to guide subsequent instruction.

To aid diagnosis, a recent development known as "Valid Confidence Testing" (VCT) will be incorporated in the current IDM implementation. (The bases for these procedures have been described by Shuford, Albert, and Massengill, 13.) With this approach, the student answers each question by expressing his degree of confidence in the correctness (on a scale from zero to 100%) of each of the alternatives, if the question is multiple choice, or if the question is an open-ended one, in the correctness of the answer he has written. Since the obtained total score is not directly related to the number of correct answers, it is in the student's best interest, in this case, to reveal the true state of his confidence; his best strategy is to maximize his subject-matter mastery rather than his score. Incorporating an adaptation of this technique in the IDM provides a more sensitive diagnostic probe of the student's knowledge space.

In selecting an appropriate strategy, it is initially reasonable to weight this factor in the developing student image in accordance with the degree of experience the student has had with the specific course material and with the Valid Confidence Testing technique. At the beginning of instruction, a student's confidence estimates will be given little weight, since he has little or no information about the course at hand. However, as he becomes more and more familiar with the course materials, the weighting of this factor will be increased. Empirical support for the VCT approach has already been obtained by Shuford and Massengill in their success within school systems in the Boston area. Both students and teachers indicated that the technique promotes rational study behavior. Preliminary experience with Project IMPACT's students is similar.

Empirical Study of First IDM

Fundamentally, the exploration of the weighting of decision factors in the IDM is concerned with the ability to predict performances (problem solutions) at end-of-course proficiency testing. The initial developmental IDM will be a realistic, simple, empirical attempt. A very limited number of factors will be used for determining decisions during this cycle, with the remaining factors used only for their correlational value. Relationships will be examined between certain characteristics of students and their performances within course levels and on the various criterion tests of performance requirements. By the nature of the student's tasks in the problem-solving COBOL course,[1] the performance requirements are for writing and debugging increasingly complex COBOL programs.

Preliminary testing of the initial COBOL course material was scheduled during the summer of 1968. The objective of this testing was to have the students generate cues for potential branching materials of differing levels of difficulty which are then to be used in the later model testing. This information was to be developed in a non-automated environment via question-answer exchanges with a first group of 12 students. All instruction was to be recorded and subsequently edited for use in the automated testing of the first iteration of the model (i.e., the first of a long series of tests of the developing model). Student glossary (INFO) development is proceeding in a like manner.

---

[1] A detailed description of the students' tasks is presented in Chapter 4

During the first iteration, the instructional decisions will be based on the state of the student's knowledge as measured by our adaptation of the Shuford-Massengill Valid Confidence Testing technique. Use will be made of recent findings regarding the varying value in different circumstances of "stimulus support," that is, confirmation or prompting (Seidel and Hunter, 1). Such stimulus support will be given each student in a manner depending upon his state of knowledge as shown by the confidence testing. For purposes of the initial model iteration, six states of knowledge will be defined; these can be expanded or collapsed if obtained data indicate this to be desirable. Figure 6 illustrates the plan for open-ended (constructed) responding. The levels are:

State One (Well Informed) will be operationally defined as a fully correct answer for an item with 80%-100% confidence. In this state, the most accelerated and difficult (highest) options of material will be presented for the next instructional material, and no stimulus support with respect to the material will be provided (no prompting or confirmation of the problem solution).

State Two (Informed) is defined as giving a highest confidence of 55%-79% for a fully correct answer. In this case, the individual will not be shifted to a higher or lower option and will not be given any stimulus support.

State Three (Partially Informed) is operationally defined by an incorrect answer which he feels is incorrect (i.e. less than 46% confidence). Both of these reflect some partial knowledge of the concepts being tested. In this case, the subject will not be shifted to higher or lower options, but stimulus support will be provided.

State Four (Uninformed) is defined as a correct or incorrect answer with 46%-54% confidence, or a correct answer with less than 46% confidence. In this case the individual is given "confirmation" support regarding the problem solution and his progress is decelerated to option-U, a simpler form of the materials.

State Five (Misinformed) is defined as giving an incorrect answer with the 55%-79% confidence assigned to the answer. In this state, the individual is decelerated to option-M, which incorporates remedial materials (different from option-U), and the individual is given confirmation or prompting stimulus support. ("Remedial" may mean reducing the information the student must handle, thereby making the material simpler, rather than repetitious presentation of the same material.)

State Six (Highly Misinformed) is defined as giving an incorrect answer with 80%-100% confidence. This will also result during the first iteration of the model in deceleration to option-M with support.

The other factors used only for correlational purposes during the first IDM iteration will involve the student's use of the glossary, the generating of his unique glossary, the information on intellectual functioning at various stages of learning deriving from an adaptation of the Guilford and Bunderson techniques, information on subject-matter structure, and finally the the various historical data (i.e., error rate and patterns, response latencies, etc.).

ITERATIVE ELABORATION OF IDM

The basic development strategy of IMPACT calls for the testing of a series of IDM versions. We have already referred to a "first iteration" of the IDM and to its empirical study, the first in the series. Beginning with a very small number of decision factors, simple decision rules, and—for lack of completed facilities—highly restricted decision options, the IDM will be progressively expanded and its sophistication increased. This will occur as a concomitant of selectively increasing the number of decision factors, elaborating the decision rules, and having more and more decision options. Each successive version of the IDM will be tested empirically in order to diagnose and isolate the most appropriate combinations of elements for the next version.

In the second iteration of the IDM, those empirical data that will have been obtained as a purely correlational output of the first iteration will be shifted to

Valid Confidence Scoring of Constructed Responses

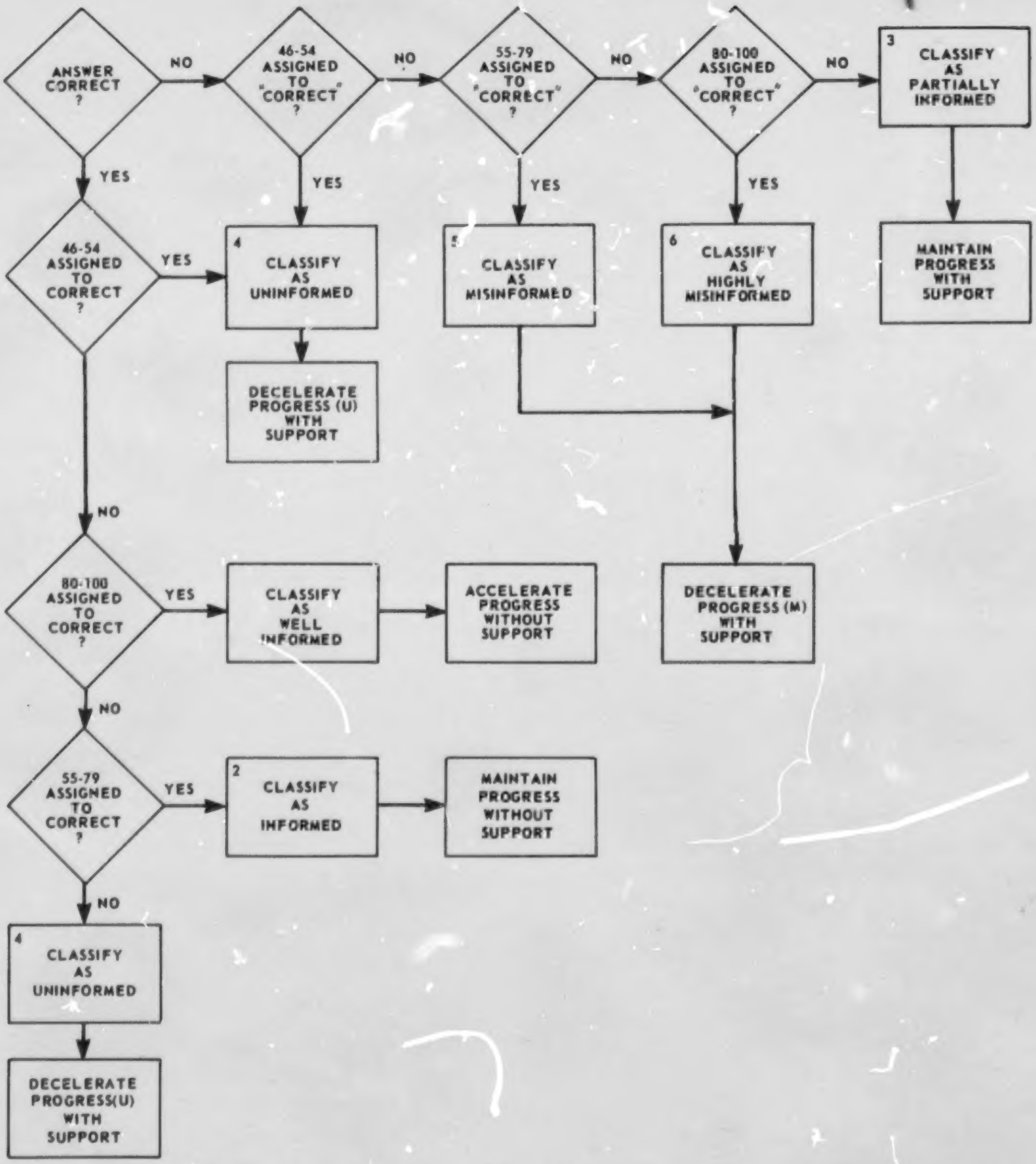

ANSWER CORRECT ?
NO
46-54 ASSIGNED TO "CORRECT" ?
NO
55-79 ASSIGNED TO "CORRECT" ?
NO
80-100 ASSIGNED TO "CORRECT" ?
NO
3 CLASSIFY AS PARTIALLY INFORMED
YES
YES
YES
YES
46-54 ASSIGNED TO CORRECT ?
YES
4 CLASSIFY AS UNINFORMED
5 CLASSIFY AS MISINFORMED
6 CLASSIFY AS HIGHLY MISINFORMED
MAINTAIN PROGRESS WITH SUPPORT
DECELERATE PROGRESS (U) WITH SUPPORT
NO
80-100 ASSIGNED TO CORRECT ?
YES
CLASSIFY AS WELL INFORMED
ACCELERATE PROGRESS WITHOUT SUPPORT
DECELERATE PROGRESS (M) WITH SUPPORT
NO
55-79 ASSIGNED TO CORRECT ?
YES
2 CLASSIFY AS INFORMED
MAINTAIN PROGRESS WITHOUT SUPPORT
NO
4 CLASSIFY AS UNINFORMED
DECELERATE PROGRESS(U) WITH SUPPORT

Figure 6

input and now used as decision factors. Preliminary weightings will be assigned to the Guilford-Bunderson structure of intellect data and to the course-historical data deriving from Valid Confidence Testing, with this assignment of weighting serving as a dynamic, rather than static, factor in the development of the next IDM version.

Traditional psychometric practice has assigned fixed weightings to given test or test battery scores (often in terms of factor loadings) used to predict success on a criterion test. In these cases the purpose is *selection* (i.e., elimination of those least likely to succeed). Within the IDM the purpose is to adjust the character of the instructional presentation *continually* throughout a course, to the optimal compromise between the kind of intellectual functioning demanded by the subject matter and the degree to which the requisite abilities are possessed by the individual student. In order to achieve this, the relative weightings given to specific abilities must also be continually readjusted to conform to the actual demands for them that are being made *at a given time*.

Correlational data from the first IDM iteration will help to establish the shifting requirements for intellectual functioning imposed by COBOL course material. However, for this approach to be used, each prospective student has to take a large test battery so that his personal configuration of abilities can be known—and this pretesting is expensive and time-consuming. The possibilities of reducing, or even eliminating, pretesting will be explored. In essence, if "loadings" of the items on each factor in Guilford's structure of intellect model can be established, each presented substantive instructional display could serve simultaneously as a test item. In effect, these tests would be administered sequentially, and, over time, the IDM would build up an increasingly precise and accurate image of any given student's ability structure.

The possibility of this sequential "testing" serves to illustrate the necessity for a statistical capability within the IDM. With sequential testing the image is compiled statistically over time; however, unless the IDM can compute the relevant statistical indices, it cannot test sequentially.

A statistical capability for the IDM is now being developed. Initially, this will be a modification of existing packages consisting of such techniques as factor analysis, correlations, cross tabulations, discriminant analyses, and so forth. Primarily, it will consist of a large number of computational subroutines that may be linked in various ways. In an *ultimate* version, this package of subroutines will be available to the operating IDM for on-line sequential diagnosis and presentation of materials for the individual student and in real time. The IDM will need only to specify the statistical hypothesis it wishes to test and the relevant parameters; the appropriate subroutines will then be selected automatically and a resultant probability will be stated to the IDM as a decision index (factor). Interim versions of the statistical capability will be limited in number of subroutines, and will not operate in real time or even on-line. However, the growth of the statistical capability clearly enhances and expedites the possibilities for testing the IDM, and it will constitute a by-product for application in general use.

With respect to subject matter, efforts will be continued to devise mathematical techniques for characterizing its structure. For example, existing syntactic charts of COBOL will be brought up to date and will serve, along with other analyses of COBOL concepts, as another benchmark for further evolving n-dimensional graphs of COBOL structure. Concurrently, efforts to devise a metric by which distances in this structure can be established will continue.

Success in both efforts implies, among other things, the ability to select for each student the most efficient path through the structure. It also implies a capability for a precise matching of the relevant portion of the instructor's knowledge space (re COBOL course requirements) and the student's, so as to detect discrepancies. At another level, this goal will be pursued via a more refined use of the glossary technique (see above and Chapter 5) aimed at probing the student's knowledge space more sensitively. It is primarily this probing approach that will be incorporated in the iterations of the IDM that will take place over the next year.

Mention must be made of one key theoretical effort that is in prospect, evolving from initial work with the student image. Complementary approaches to the student image are suggested by the work of Guilford and of Gagné. Theoretical work will be undertaken to attempt to formalize and develop the relationships between the how of intellectual functioning (e.g., Guilford) with identifiable units of informational requirements, or the what of such functioning (e.g., Gagné).

Chapter 3

DESCRIPTION OF THE HARDWARE BASE OF THE IMPACT SYSTEM[1]

HARDWARE SUBSYSTEMS

For purposes of discussion, the IMPACT hardware base may be divided into three major subsystems:

(1) Information Processing Subsystem
(2) Data Storage Subsystem
(3) Communication Subsystem

The three subsystems are described below and are schematized in a chart of the HumRRO hardware configuration (Figure 7).

Information Processing Subsystem

This phrase applies, in essence, to the central processing unit (CPU), which furnishes the information processing capability for the Instructional Decision Model and for various other purposes. The selection of the CPU was dictated largely by the fact that an IBM 360/40 computer system was being delivered to HumRRO as IMPACT was initiated. In part, the decision to order that system had been influenced by the consideration that it might serve both the information processing requirements of HumRRO at large and of a then highly restricted CAI effort. The CPU originally delivered contained 65K of core; this core capacity was increased to 128K in September 1967 and is currently being increased to 256K bytes.

Data Storage Subsystem

For permanent storage, four IBM 2401-5 tape drives are available. These tape drives have a relatively high speed and a relatively high data packing density. For IMPACT, their primary function is that of permanent data recording. Operating systems of various types, instructional content, and other types of information to which rapid random access is required are resident on disk packs. Originally four IBM 2311 disk drives were used; they are being replaced with nine IBM 2314 disk drives.

Communications Subsystem

This is the system by which students communicate with the instructional decision model resident within the CPU. For computer-administered instruction it may be viewed as the interface between machine and student, and is therefore of great importance. For initial development purposes, there are two essentially different versions of the communications subsystem.

---

[1] Identification of products is for research documentation purposes only; this listing does not constitute an official endorsement by either HumRRO or the Department of the Army.

HumRRO Hardware Configuration

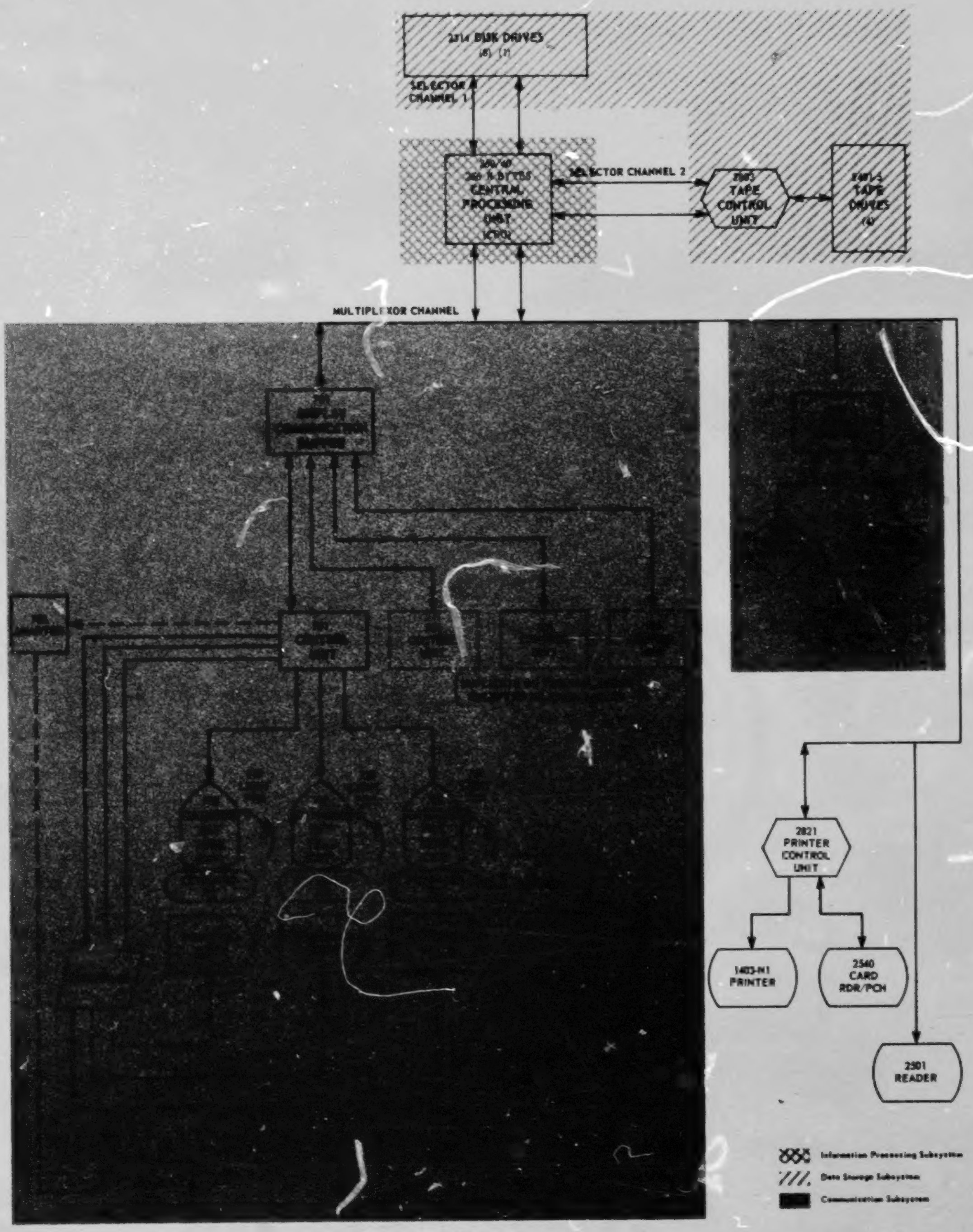

2314 DISK DRIVES
SELECTOR CHANNEL 1
CENTRAL PROCESSING UNIT (CPU)
SELECTOR CHANNEL 2
TAPE CONTROL UNIT
TAPE DRIVES
MULTIPLEXOR CHANNEL
2821 PRINTER CONTROL UNIT
1403-N1 PRINTER
2540 CARD RDR/PCH
2501 READER
Information Processing Subsystem
Data Storage Subsystem
Communication Subsystem

Figure 7

29

The first and interim type of communications subsystem involves the use of IBM 1050 typewriter terminals through the IBM 2701 Data Adapter. Three of these terminals are currently in use. The first is used primarily for software development and for debugging. The second terminal, which incorporates the visual feature (i.e., a random access Carousel projector), is being used to convert to CAI format, a preexisting programed instructional course in basic computer programing (Seidel and Hunter, 1968, 1). A third terminal is located remotely at HumRRO's Division No. 3 in Monterey, California, and is being used to develop the IMPACT statistical capability (see Chapters 1, 2, and 6).

The major communication subsystem has as its core the Sanders Associates 720 Data Display System. A total of 12 student stations will be built initially and two hard copy (teletypewriter) facilities will be provided for these 12 stations. The 12 stations operating in Alexandria, Virginia, as well as the West Coast station, will operate in the remote mode so that the system will accommodate student terminals at any geographic location having standard telephone lines. This means that the computer software being developed for the in-house capability will not have to be revised for remote field operations.

The Sanders 720 system consists of the 731 Display Communication Buffer, the 701 Control Unit, the 708 Cathode Ray Tube (CRT) display, and the 722 Keyboard together with the 737 Light Pen. The student has immediate contact with only the last three items (708, 722, 737). Printed messages appear on the cathode ray tube face. The messages may be generated either from the keyboard by a student, by a course author, or by the program resident in the CPU. Responses may be made either via the keyboard or via the light pen (which has passive recognition capability only).

Each 708 CRT will be augmented with an auxiliary visual display. In effect, this will be a rear projecting screen onto which images will be projected by a Perceptoscope, a type of 16mm projector that will be controlled from the 701 Control Unit through a special interface and digital device controller.

The Perceptoscope is capable of displaying 16mm film (silent only) at any of 18 speeds from 0 to 24 frames per second. A single frame of film may be selected and held on the screen indefinitely without damage to the film or diminution of the transmitted light. Films may be moved either forward or backward at a maximum speed of 24 frames per second, so that at any time a block of 48 frames is available for a semi-random access within a period of one second. Brief motion sequences may also be shown. A unique feature is the existence of a second film. While the back film cannot be focused simultaneously with the front film, it can be used to obscure portions of the front film, to highlight particular areas, to provide curtain effects, and so forth. The back film is controllable independently of the front film. All film movement in the Perceptoscopes will be entirely under program (IDM) control.

DEVELOPMENTAL AND EXPERIMENTAL STATIONS

The 12 planned stations will be divided into 10 developmental and two experimental stations. The developmental stations will be equipped as described above. The experimental stations will contain, in addition, special features whose desirability will be evaluated to establish whether their general adoption (i.e., in the remaining stations) appears warranted.

For example, one of these experimental stations will be equipped with the Sylvania Tablet, an electronically sensitized sheet of glass in a metal frame over which paper may be placed. The student writes on this paper, using a ball point pen that is electronically connected to an analogue to digital (A/D) converter, which provides a digital output to an interface incorporating character recognition software. In other words, so long as the user follows certain minimal conventions in printing capital letters and Arabic numbers they will be recognized by the equipment.

Thus the Sylvania Tablet provides an alternate input to the keyboard. It will enable student COBOL programers to write directly on COBOL coding sheets much as they will do later on the job. Moreover, images may be projected by the Perceptoscope directly onto a screen underneath the transparent Sylvania Tablet so that, with proper registration, identification of portions of the projected images, or even tracings of paths within the images can be recognized.

For the future it is hoped that it will be possible to test within the experimental stations audio-output and voice-input features. Another possiblity being considered for the experimental stations is a closed circuit TV capability.

Chapter 4

COBOL COURSE DEVELOPMENT I: OBJECTIVES

INTRODUCTION

One stated aim of Project IMPACT is to develop a course to teach COBOL computer programing specifically oriented toward training for the Army data processing jobs. This chapter will deal with (a) the rationale behind the development of the training objectives; (b) the characteristics of the job focused on for IMPACT course construction; and (c) the attributes of the potential trainee population. The chapter following this one will describe the development of the instructional content for the course.

A succinct description of COBOL is found in its name, an acronym formed from the phrase COmmon Business Oriented Language. Its "commonness" derives from the fact that COBOL was designed to run on practically any computer, given a few minor language or system modifications; in other words COBOL is "machine-independent".

The language is "business oriented" in two respects. First, COBOL's functional capabilities are oriented toward the data-processing operations performed in business and industry, such as record-keeping, account up-dating, and file maintenance. It is distinct from algorithmic programing languages such as FORTRAN, which is designed for such purposes as the solution of complex equations or the analysis of experimental statistics.

The other aspect of COBOL's business orientation stems from its designers' wish to make it maximally understandable to management. To accomplish this goal, the sentences or statements of COBOL have been constructed to conform as closely as possible to ordinary English. For example, the statement "OPEN INPUT SALES-FILE, OUTPUT BILLING-FILE" is easily read as analogous to the business operations of pulling file folders on sales and billing preparatory to updating a customer's account.

ESTABLISHMENT OF TRAINING OBJECTIVES

In developing training objectives for the course, the following groups of questions had to be answered by project analysts:

(1) Who will be taking the course: What is to be expected in terms of prior experience, education, aptitude, and attitudes on the part of the trainees?

(2) What are the tasks the trainees will be expected to perform on the job? Which of these tasks are suited to formal, as opposed to on-the-job, training? Which of the tasks should take priority—that is, which parts of the course should be developed first?

(3) What criteria should be used to evaluate proficiency of the trainees? What should be the standards for determining whether training objectives have been met?

Two other related questions are involved as well:

(4) What channels could be used in getting regular feedback from the Army on the value of the course?

(5) How will the COBOL course fit into overall Automatic Data Processing (ADP) training programs found in the Army?

Answers to these groups of questions were determined by collecting data from various Army sources. Findings are presented in the remainder of this chapter. Because the field of data processing is continually changing, the information presented can be considered current as of the third quarter of FY 1968. The analysis had to be terminated during that period in order to fit the results into the overall development schedule of IMPACT. A further qualification to the representativeness of the data lies in the fact that in many instances appropriate information was not readily available in meaningful form and inferences had to be made from secondary sources.

Tasks

Information on tasks performed, relative importance of the tasks, and tasks that are suited to formal training was obtained by project analysts by telephone and personal interviews from the following sources:

1. Programers and programer-analysts at Army data processing installations.
2. Programer and analyst supervisors at Army data processing installations.
3. Data processing managers.
4. Documents, cited in the text of this paper.

For initial course development attention was focused on the programers, although eventually it is planned to develop course modules appropriate for supervisors, managers, and operators.

It became quite apparent as contacts were developed at various ADP installations that extreme heterogeneity of jobs and job structures existed from base to base. Thus, any overall generalizability of COBOL course modules could arise from only two sources: (a) sampling representatively the types of programing problems that are worked on at the various ADP installations and (b) restricting the initial course modules to "third generation" computing configurations and concepts (i.e., COBOL 360).

An IBM 360 is a type of computer referred to as third-generation equipment; the difference between it and earlier machines (first and second generation) lies in more efficient design of core storage, the introduction of more intrinsic functions (i.e., capabilities now a part of the computer, that can be called upon rather than written by the programer), and a more efficient coupling of peripheral devices (e.g., disk drives) with the computer itself.

In general, third-generation equipment can perform a given task more efficiently than second generation, provided the programer and systems analyst are aware of and can utilize the improved capabilities. For the analyst it is necessary to acquire a general knowledge of the system's capabilities plus some knowledge of the language (or languages) used; the programer must not only learn these capabilities (and in more detail) but also must learn or update his knowledge of the programing language (or languages) in use on the system. In COBOL, the advent of third-generation equipment caused a considerable increase in the number and variety of language "macros" (a macro is essentially a group of commands that accomplish in one word what first- and second-generation equipment did as subroutines).

Terminal Objectives

Terminal objectives, (sometimes called end-of-course objectives), are statements of what the student is to be able to do (behavior) upon completion of the course. Tests to be administered at the end of the course are derived from this list of objectives, and the content and organization of the course itself are in part derived from the statement of terminal objectives.[1] Naturally the student must achieve many subobjectives during the course.

Terminal objectives are from time to time re-examined by Project IMPACT personnel and may be revised in accordance with new information. The objectives now being used are as follows:

Objective I: Program Writing

Given the types of specifications listed below, the student writes in each case a program to produce the specified output. The student has access to COBOL programing manuals and the definitions of terms he has learned in his course. No constraints are placed on time to produce the program, or number of compilations, or efficiency of object code generated. The student must also prepare Job Control Language cards and operator instructions needed to run his programs. The student also prepares program documentation. (Note that the specifications described below are derived from several actual Army data processing programs.)

Specification 1: Personnel Accounting

This program involves one input file and one output file. The problem is to extract information based on three conditions. The layout of the input and output records is defined in detail in the specification. All data exceptions must be handled by the program, with explanatory error messages. But the ease of maintenance and of future modifications is not considered.

Specification 2: Personnel Accounting

This program involves editing input file with one and two output files. The criteria for editing are based on data in pre-stored tables. The layouts for input and output records are defined in detail in the specifications. The program is to be organized for ease of modification of editing criteria.

Specification 3: Supply

This program updates a file involving two input files and one output file (the updated master). The input files have already been edited and sorted into proper sequence.

Specification 4: Operations

This program involves solving algebraic equations, storing intermediate results in tables, and processing a printed report of computation results.

Objective II: Style

1. The student is asked to list aspects of programing style which affect readability and modifiability of programs, and lists all of the following:

   (a) Meaningful data and procedure names.

   (b) Frequent comments.

---

[1] Terminal objectives do not always consist of *all* job behaviors. Some skills are acquired most effectively through on-the-job training or practice.

(c) Simple (as opposed to complex) compound COBOL statements.

(d) Program organized into separate logical subroutines or modules.

2. When the student is given a previously written "procedure division," which does not meet any of the above criteria, he can identify all the violations and make appropriate changes.

Objective III: Debugging

1. The student, when asked, can list all the sources of debugging information shown below:

(a) Compiler diagnostics.

(b) System messages when the program hangs up.

(c) Output of a TRACE.

(d) Printout of output data.

(e) Printout of input data.

(f) Explanations in the reference manual of compiler diagnostics.

(g) Information in the reference manual, on language requirements.

2. Given detailed file layouts and a pre-written data division, the student can locate and correct all the errors in file and record descriptions.

3. Given the specifications for a program and a pre-written procedure division the student can locate and correct all of the errors in syntax and logic.

4. The student can locate and correct errors in his own programs (from Objective I). The standard for his performance is error-free output.

Objective IV: Using Reference Materials

Given the reference manual on COBOL (IBM DOS COBOL Reference manual), the student can answer correctly the majority of ten multiple-choice questions on COBOL elements not taught in the course in the 30 minutes.

Objective V: Documentation Standards

For each program listed in Objective I, the student completes documentation according to certain standards. (These standards will be derived from existing Army data processing installations standards.)

Objective VI: Modifications

Given a change in the editing criteria for Specification 2 above, the student can modify his program to meet the changed specification.

CHARACTERISTICS OF THE JOB

COBOL in the Army

Nine Army installations have been visited (1 July 1967 - 30 June 1968) to find out how COBOL is actually used, and to determine the major tasks performed by COBOL programers and analysts. A number of representative Army ADP installations were able to provide program specifications and program listings. Project IMPACT analysts interviewed 36 programers, analysts, trainees, and data processing supervisors at the installations visited, and additional information was collected through telephone interviews with personnel at eight more installations. Thus, a total of 17 major Army ADP facilities were contacted.

Initially, the plan was to develop three separate sets of job descriptions (for programers; for analysts, supervisors, and managers; and for computer operators), with the course then containing separate "modules" of instruction, depending on the trainee's objectives. However, as information was collected on the tasks performed by personnel at the various installations, it became apparent that these jobs vary from installation to installation, depending primarily on the following factors:

(1) Size of the installation in numbers of personnel.
(2) Type of computer hardware used.
(3) Programing languages used.
(4) Type of applications (personnel, accounting, supply, etc.).

Hence the decision was made to focus primarily on programers.

Since computer technology is changing rapidly, Army data processing installations are in different stages of technological development. There are major Army efforts under way to standardize some of the applications that apply to many installations, such as NAPALM (National ADP Program for AMC Logistics Management) and COCOAS (CONARC Class One Automated Systems). Such large integrated systems, with their centrally written and maintained programs, are another factor leading to change in the job of an installation programer. This is one facet of the evolution toward third-generation computer systems; another is the continuing activation of computer hardware with third-generation facilities.

Given the dynamic nature of the data processing field in the Army, it is important to ensure that training objectives of any COBOL course will not become obsolete in a short time or, worse yet, teach outdated concepts from the beginning. Therefore, Project IMPACT analysts focused on:

(1) Tasks performed in installations that seem to be more advanced technologically.
(2) Tasks common to the job of the programer in all or most data processing installations, that is, tasks that are not peculiar to a particular hardware or software system.

Job Specifications of COBOL Programers

The tasks common to the job of programer in COBOL in many different Army installations, based on information collected from Army data processing installations, are:

(1) Study the specifications of the problem.
(2) Plan the logical flow of the solution.
(3) Plan for dealing with missing or erroneous input data.
(4) Write the program in COBOL and have it keypunched.
(5) Review the program cards as punched by the keypunch operator and interpreted by the interpreter.
(6) Have the program compiled; study the diagnostic messages from the compiler; correct any errors in the program cards; repeat this process until no diagnostic errors occur.
(7) Write operator instructions for running the program.
(8) Choose test data for running the program, and have the compiled program run with the test data.
(9) Diagnose errors in the program from the test runs, correct them, and recompile; repeat this process until no test run errors occur.
(10) Check to see if there are any changes in the original specifications, and if so, modify the program.
(11) Check to see whether pre-stored environment or data divisions (technical terms in COBOL) are available for the program; if so, write COBOL instructions to call them.

(12) Check to see if standardized data names are to be used for the program data; if so, use those names in the program.

(13) Prepare any Job Control Language cards (if the computer system has an automatic job scheduling system) that are required to run the program to draw on any system software facilities (sorts, merges, file formating, printing) that are to be integrated with the program.

This then comprises the job specifications for programers using COBOL at this time.

Some tasks from the above list were selected as appropriate for of end-of-course objectives. That is, they were to be taught formally as part of the COBOL course. Only those tasks directly involved in writing COBOL programs were involved. Those tasks that could be readily learned through on-the-job training and not requiring formal school training were omitted. Special emphasis was placed on (a) tasks supervisors cited as very important but often neglected or poorly performed by programers and (b) tasks cited by new trainees as being most difficult.

STANDARDS AND EVALUATION

Criteria for evaluating the proficiency of programers are not well defined in data processing organizations. Objective, quantifiable criteria are available, such as number of compilations needed, amount of time spent writing the program, or efficiency of the object code as measured by running time, or amount of storage used. However, these items are not typically used as criteria by supervisors and are not regarded by programers as important. Most supervisors stated, in effect, "Gets the job done" as the main standard. This means that the program runs, producing output satisfactory to the customer. It also means that the programer was able to handle any constraints which might apply, such as externally imposed time deadlines, limited core storage available, poorly defined specifications, or erroneous input data.

One of the installations sampled has compiled a list of the considerations involved in estimating time required to perform a programing job (Appendix B). This supervisor's check list helps define the elements involved in the job and the conditions under which the job is to be performed. Hence it provides a first step towards establishing performance standards.

CHARACTERISTICS OF THE PROGRAMER TRAINEE

The development of a course of instruction that effectively prepares the trainee for the responsibilities of a given job calls for detailed knowledge about (a) the behaviors required to perform the tasks comprising the job, and (b) the relevant characteristics of the persons to be trained for the job.

The first type of information provides the basis for the content of the course, that is, "what shall be taught." With the aid of a job description, the instructional designer prepares a set of "terminal objectives" which consist of imparting to the student those behaviors deemed necessary for job performance at some level of competence. He additionally prepares a set of "enabling objectives" which comprise teaching behaviors necessary to the attainment of the terminal objectives. For example, if a terminal objective is to teach the programing trainee to debug IBM 360 programs at the machine language level, an appropriate enabling objective would be to teach him the hexidecimal system since core dumps (which contain the necessary information) are printed out in this notation.

The second type of information, the characteristics of the trainees, is an important determiner of the strategy and tactics used in presenting the course content—that is, "how shall we teach?" A COBOL trainee with experience in another programing language may need to be taught only the COBOL conventions of conditional branching, while the novice will most certainly require preliminary instruction in the purpose and logic of this technique. The student who has difficulty handling figural and spatial representation of concepts may, during instruction on flowcharting, require considerably more verbal explanation than the student who is not limited in this way.

Both of these situations are examples of trainee "entry" characteristics relevant to the teaching of COBOL. However, the second instance represents a class of traits highly specific to the research aims of the project, and measures of them would normally be available only from psychological tests administered on-site; it is not to be expected that they would be found in the background records of potential COBOL trainees (although this might not be the case in the future if the tests are highly predictive of performance in training). Thus assessment of these variables as contributors to the manipulation of IMPACT instructional materials must await the testing of sufficient numbers of trainees to ensure the stability of whatever relationships are obtained.

The first instance, prior experience, is one of a number of entry characteristics frequently employed by industry, government, and the military in selection and training of personnel. The current efforts in Project IMPACT are directed at these variables with the expectation that sufficient data will become available on existing incumbent and trainee groups to permit inferences to potential student populations.

To familiarize the reader with the kinds of information being sought and to provide a framework for presenting the data obtained to date, the projected set of trainee entry characteristics and its organization will be presented first. For those elements of the set on which data are available, either a description or an estimate of its distribution over the population will be made, depending on the completeness of the data at the present time.

The potential trainee population is being categorized along the following dimensions:

Category 1—Job-Trainee. Programer or Systems Analyst[1]
Category 2—Military Status. Civilian or Soldier
Category 3—Rank or Grade
  (a) Civilian: GS-5 through GS-7, or GS-9 and above
  (b) Military: Enlisted or Officer
Category 4—Trainee Sources.
  (a) Military (Officers and Enlisted Men)
    (1) Enlistment or Commissioning
    (2) Draft
    (3) In-service applications
  (b) Civilian: Within or outside government
Category 5—Prior ADP Experience.
Category 6—Educational Background.
Category 7—Skill-Related Aptitudes.

Each job category is being broken down into numbers of civilians and soldiers, then further subdivided by rank or position and again by source of trainee. Further partitioning occurs on the basis of prior experience and, finally, data on vocational aptitudes and educational level would be obtained for each of the 144 "cells". The purpose of this rather extensive categorization is to provide the IMPACT staff with data sufficiently flexible to permit whatever combinations of traits become meaningful as the project develops. It would additionally provide information concerning those background groups tending not to enter the ADP field (since it is expected that some cells would contain zero or only a few entries).

---

[1] It is recognized that jobs in either one of these groups may involve functions found in the other; however, emphasis is to be placed on jobs that are primarily programing or primarily analysis.

The first three breakdowns will provide information on an important aspect of the student population: the number of civilian programers and systems analysts—trainees and professionals—employed by the Army relative to the corresponding numbers of military personnel. Since these two groups differ in selection criteria, prior ADP experience, and, to a lesser extent, the nature of the job, the training they receive in our CAI COBOL course would also differ. Which of the two groups to emphasize in the development of the course depends in large measure on knowing their relative numbers, both current and projected.

The fourth categorization, trainee source, is important in two respects. Particularly for military personnel, an individual's status as draftee, enlistee, and so forth provides a gauge of the amount of time he can be utilized by the Army, which in turn indicates the training expenditure appropriate to his service period. Thus, the relative proportions of trainees expected from each source will affect the number and complexity of skills taught in the COBOL course. Secondly, trainee source reflects to some extent the reasons for an individual's presence and his attitudinal and motivational make-up with regard to his task.

The remaining three categories—prior work experience, educational level, and skill-related aptitudes—are clearly related to the pedagogical tactics of any course of instruction, regardless of subject matter or presentation medium.

The prior experience entry characteristic, unlike the others, can be meaningfully discussed in direct relation to the subject matter now under development in Project IMPACT, that is, COBOL for the IBM 360 computer system.

An IBM 360 is a type of computer referred to as third-generation equipment; as noted earlier, third-generation equipment can perform a given task more efficiently than second-generation, provided the programer and systems analyst are aware of and can utilize the improved capabilities. The analyst must have a general knowledge of the system's capabilities plus some knowledge of the language or languages used. The programer must know the system in detail, and have knowledge of the programing language or languages in use on the system. These characteristics of "generations" of computers provide the basis for organizing background experience as follows:

(1) No ADP experience whatever or in early stages of training
(2) Programing experience
    (a) Second-generation systems
        1 COBOL
        2 No COBOL
    (b) Third-generation systems
        1 COBOL
        2 No COBOL
(3) Systems analysis experience
    (a) Second-generation systems
        1 COBOL
        2 No COBOL

(b) Third-generation systems

1 COBOL

2 No COBOL

Individuals in subcategory (1) would, naturally enough, require COBOL instruction in a context that also imparts that basic knowledge of the logic and design of computers requisite to minimally acceptable use of the language; how much of the basic knowledge is indeed requisite remains a topic for research. Subcategory (2) first contains programers with experience on second generation equipment. Those with COBOL experience would need instruction and practice in employing the third generation conventions of the language (e.g., the new macros) and the new capabilities of the hardware; those with no COBOL experience would, of course, have to learn the language completely.

Neither group, however, would require any instruction in basic programing concepts (e.g., conceptually, a loop is still a loop regardless of the programer's specific machine and/or language experience). Third-generation programers would, of course, be even better prepared to work with 360 COBOL; those who know the language are in most cases "ready to go"[1]; those who do not, will have to learn it. Both groups are presumed familiar with the general hardware features of the third-generation machines.

Systems analysts, subcategory (3), can be considered in the same manner. Their skills in conceptualizing a problem are relevant regardless of the hardware and software; what remains for second-generation analysts is to become generally familiar with the language and the computer's capabilities. Third-generation analysts who do not know the language need only to become generally familiar with it; those who have COBOL experience are, again, nearly "ready to go."

Data obtained to this time on the characteristics of the trainee population are briefly summarized below:

Category 1 - Job Trainee. The most recent information available indicates that the Army employes approximately 2,800 programers and approximately 2,000 analysts.

Category 2 - Military Status. Approximately two-thirds of the programers (about 1,900) are civilian personnel. Similarly, about two-thirds of the analysts (over 1,300) are civilians. A third job category of civilian ADP personnel encountered in the survey is that of Computer Specialist, of whom there are about 650; more data will be gathered on this position.

Category 3 - Rank or Grade. About one-third of the civilian programers are in grades GS-5 or GS-7, in training as ADP Interns; the remainder, mainly GS-9 through GS-11, function largely as programers although they also have some administrative duties. Approximately 300 hold higher-ranking specialized or managerial positions.

Little information is yet available for the military personnel. There are about 1,000 military programers and assistant systems analysts, all of whom are enlisted men. All military systems analysts are officers.

Category 4 - Trainee Source. The only information yet available has come from interviews and correspondence with several data processing installations. This category of information will be expanded as more data become available from such sources as the U.S. Army Data Support Command.

---

[1] One qualification must be made; while all third-generation computers exhibit the same sorts of features, they have varying idiosyncrasies. A programer or analyst going from one make of hardware to another will have to learn these quirks and how they affect system performance and program writing.

Category 5 - Prior ADP Experience. The findings in the several experience subgroups vary in representativeness; most of the present data come from telephone interviews with a sample of 15 data processing installations, to be followed by more thorough surveys. Twelve of the 15 installations surveyed have second-generation equipment; nine of the 12 use COBOL, and all of their programers and analysts have COBOL experience. At the three third-generation installations, all of which use COBOL, 88% of the civilian and all of the military programers had COBOL experience.

Category 6 - Educational Background. No information is currently available on the distribution of educational background over the Army ADP employee population. For working purposes, information on the minimal educational requirements for admission to training and similar indirect sources on educational background of military ADP personnel and related civilian occupations are being used.

Category 7 - Skill-Related Aptitudes. Here also, no data concerning distribution of aptitudes are yet available; information on the pertinent aptitude tests and minimal qualifying scores are being used in planning operations.

Chapter 5

COBOL COURSE DEVELOPMENT II: INSTRUCTIONAL CONTENT

INTRODUCTION

The development of a COBOL course is one of the objectives of the first cycle of Project IMPACT. The Instructional Programing staff has begun developing preliminary course content (hereafter called the Preliminary COBOL course). The primary basis for course development was a survey of Army ADP installations.

The development of the ultimate course will be based not only on the objectives resulting from the Army survey, but also on the actual testing and revision of preliminary materials in a controlled situation, on groups of students whose entry characteristics have been measured and whose performance is monitored.

During the year of development work ending 30 June 1968, several types of activities entered into the selection of material for the first data collection sessions with students, which were scheduled for August 1968:

(1) After establishment of course objectives, the director of Instructional Programing trained the staff in COBOL.

(2) Each instructional programer developed, on the basis of his COBOL-learning experience, a technique for teaching COBOL in a tutorial setting. Each instructional programer taught at least one student.

(3) On the basis of the tutorials, the Instructional Programing team developed an outline of the basic elements of COBOL, establishing the behavioral objectives required for writing simple COBOL programs.

(4) On the basis of the tutorials, the Instructional Programing team developed an outline for the content and organization of the Preliminary COBOL Course.

(5) The Preliminary COBOL Course was prepared. Tutorial students were used at each rough-draft stage as an aid in refining the content.

(6) Independently, criterion tests were written for the various levels of the course (these levels are described below). The criterion tests are administered to students at the various rough-draft stages of development.

The Preliminary COBOL Course was to be tested with 12 students in July-August 1968. Subsequent instructional content activities were planned as follows:

(1) On the basis of the Preliminary Course test, the course will be modified and additional material will be added to provide for greater individualization of instruction.

(2) The expanded course will be programed to be administered by computer, incorporating the other features of the system described elsewhere in this report (response analysis, data collection, decision rules, the various input-output media, etc.).

(3) The course will be tested at HumRRO in Alexandria, Virginia, in its computer-administered form, with at least 50 students. It will then be evaluated, modified, and retested with a larger sample both at HumRRO and in the field.

This chapter first outlines the organizational rationale for the content (the interactive procedures of the Preliminary COBOL Course are illustrated in Appendix C). The criterion test development is then described. Finally, the specifics of the planned response analysis will be given, including overall design considerations of the preliminary formating as well as plans for revision following initial data collection during July-August 1968.

OUTLINE OF CONTENT AND ORGANIZATION

One of the most important concepts underlying the Preliminary COBOL Course organization and content is that of functional context. The following summary of the approach is taken from Shoemaker's (14) The Functional Context Method of Instruction:

> The main feature of the functional context method is its advocacy of a topic sequence wherein the *functional significance* of each topic is firmly established for the learner prior to, and as a context for, the learning of novel and more detailed material relating to the topic. Two requirements govern the choice of the learning context for any topic. First, the context must have significance for the learner, i.e., it must be meaningful to him on the basis of previous learning in the course and ultimately his pre-course experence. Second, a context must have functional significance, i.e., it must be directly relevant to the goals of the instruction.

The functional context method of instruction has been demonstrated to be effective in the field of electronics maintenance (15). Applied to COBOL language programing, it means that the language is taught in the context of overall programing logic and Army data processing problem-solving tasks.

The Preliminary COBOL Course is organized into levels. The term "level" is used to suggest a hierarchy of skills, in the context of COBOL program writing. Each level is designed to achieve behavioral objectives that are prerequisite to the next level and to attaining proficiency in the terminal objectives discussed in Chapter 4 (and implemented by the criterion tests described later in this chapter).

At the beginning of each level, a typical Army data processing problem is presented. The material taught in that level is presented in the context of solving the problem, and only information which the student needs to solve the problem is presented. If a student wants to know more about a topic than the authors thought was needed at that point in the problem solution, he can ask for INFO. For example, the student might feel he needs to know more about how the card reader works before he uses the COBOL READ statement.

A request for INFO is made by the student entering the word INFO, followed by the word to be defined. This gives him access to the course glossary, which contains such material as definitions (other than the short definitions that appear with the text), diagrams, and samples of COBOL coding formats. For comprehensive displays the student is referred to the auxiliary visual device such as the Perceptoscope screen. The instructional programers stipulate the terms to be included in the glossary by listing, for each text display they write, the terms that are being introduced for the first time.

Each request by a student is to be "tagged" so that an individual record will be maintained for each student. This unique history will provide a further probe of the student's organization of the presented material. The various records will be used correlationally to upgrade subsequent iterations of the IDM. Student requests for terms not included in the glossary will also be used to update and revise the glossary and to provide input to IDM development. Thus, the information gathered from the use of the glossary in the preliminary course will contribute to the data gathered on student performance as well as to the final determination of the structure of the course.

As the student progresses from one level to the next, the problems increase in complexity. Hence the concepts and skills he learned earlier are further refined, and new concepts needed are introduced. In this way, emphasis is placed on solving problems, developing skills, and learning to use new tools, rather than memorizing rules and components of the COBOL language.

A flowchart describing student-system interaction for Level 2 is presented in Appendix C. This flowchart is designed solely to illustrate the content and organization of the preliminary course. Since the course is continually revised and improved on the basis of data collected from test students, the flowchart is not a completely accurate representation throughout. The flowchart also varies in level of detail. At some points, a detailed description of student-system interaction is shown, to illustrate various techniques. At other points, only the general topic is indicated.

Not all of the course logic is shown in the flowchart. For example, positive feedback is indicated after each correct student response. Sometimes this feedback is simply a comment such as "good," "well done," or "exactly right." These comments are to be programed as a course option. That is, by turning off a switch, an experimenter may, for selected students, delete those comments from the course.

TESTS OF PROFICIENCY

Criterion Tests

Criterion tests are being developed for each of the four levels of the Preliminary COBOL Course. Each of these criterion tests will consist of two parts: (a) an on-line test consisting of multiple-choice and constructed-response test items; and (b) an off-line test requiring the student to write a program.

For the off-line program writing subtest, the student receives written program specifications appropriate to the level of the course he has just completed. His task is to write a program and have it run successfully from these specifications. The student has access to reference materials he might need to help him write the program.

The on-line multiple-choice, constructed-response subtest focuses on programing concepts, specific programing techniques, COBOL vocabulary, program logic, and flowcharting as well as on segments of program writing. It seeks to measure the student's ability to deal with the many elements that make up a program, while the off-line test focuses on his ability to combine these elements into a functioning program.

On-Line Multiple-Choice, Constructed-Response Subtests

Presently, all test items are being designed for presentation to the student via the Cathode Ray Tube (CRT), the Perceptoscope screen, or a combination of the two media. At a later date, presentation of some of the test items will be adapted to utilize the Perceptoscope projection on the Sylvania Tablet capability.

Test items currently require student response via either the light pen or the CRT keyboard. When the Sylvania Tablet becomes available, a number of the keyboard response test items will be converted to tablet response.

The multiple-choice, constructed-response subtests for Levels 1 and 2 have been constructed, tested on students, and revised. Revisions of format, content, wording, and response form have all taken place. A brief description of these two tests appears below. Development of criterion subtests for subsequent levels is under way. The Level 1 and Level 2 subtests were to be administered to students during the Summer 1968 COBOL Course.

The Level 1 on-line criterion subtest consists of 24 items, of which 18 are multiple choice and six constructed response. The glossary option is inoperative during the test and no constraints are imposed on response time. The estimated average on-line testing time is 15 minutes. A short description of each test item and its course concept reference appears in Appendix D.

The Level 2 on-line criterion subtest consists of 23 items—16 multiple choice and seven constructed response. The glossary option is inoperative during the first part of the test. During the second part of the test, the glossary is operative and the student can use INFO to request any reference material he feels he needs to enable him to answer a test item correctly. There are no constraints placed on response time. The estimated average on-line testing time for this subtest is 25 minutes. Appendix Table D-2 gives a short description and course concept reference for each item in this test.

RESPONSE ANALYSIS AND DATA COLLECTION IN PRELIMINARY COBOL COURSE

Several purposes will be served by collecting data on student responses in the Preliminary COBOL Course. The data collection system to be described here is designed to meet the needs of the following users:

Course Authors
- (1) To evaluate the instructional content.
- (2) To revise the content.
- (3) To plan for alternative methods of presentation.
- (4) To evaluate and revise criterion tests.
- (5) To develop feedback displays based on actual student performance.

Behavioral Scientists
- (1) To relate pretest factors, learning behavior, and criterion performance.

Coursewriter Programing Staff
- (1) To provide further response analysis requirements.
- (2) To determine hardware and software requirements.
- (3) To improve the data collection system itself.

Definitions

Response is defined as the contents of the input buffer in the computer, after a student has transmitted a message to the computer. With the Sanders CRT, this means that the student has pressed the SEND button which causes the message to be sent from the CRT buffer to the computer input buffer. Any student-initiated SEND is considered a response even though it may consist of only the SEND code with no text.

Response analysis is used to mean the computer-programed techniques and conventions for recognizing, classifying, or interpreting the student response or a sequence of student responses.

Data collection is used to refer to the conventions and techniques for storing information about student responses.

Data available on-line in this context means data stored on random access disk or in core storage, in a form accessible to the Coursewriter system while Coursewriter is servicing student stations (Chapter 6 contains further information on Coursewriter).

Off-line processing refers to either manual or computer processing of data which is not part of the Coursewriter system operations while Coursewriter is servicing student stations.

General Design Considerations

The task was to design a system for response analysis and data collection that would provide the required information in a form most easily used for all the above purposes. In most cases, "easily used" means that the information is available to the user in a specific, already interpreted form. Data that could not be easily used by any of the groups, for example, would be a verbatim list, in order of input, of all students' responses throughout the course.

On the other hand, to develop a system of highly interpreted and classified data collection would mean that a number of a priori assumptions would have to be made about the anticipated student responses. For example, an attempt could be made to choose five or six categories that seemed to reflect the major concepts involved in the subject matter being taught, or five or six categories that seemed to reflect the major information processing activities the student would be performing. All responses could then be classified according to these categories and the interpretation provided along with the data collection system. However, to prejudge the data in such a gross way would clearly not be an acceptable approach. The system described here is an attempt to find a middle ground, between two extremes, for clustering responses in ways and to a degree that will be both meaningful and efficient.

In the preliminary course, decisions about alternative presentations to the student (feedback, review, etc.) are based only on the preceding response and/or specific student request. At this stage of development the decisions are not based on response patterns or summary data or pretest data. In fact, the main purpose in the preliminary course is to collect data that will provide a basis for making these more complex instructional decisions. Hence the data collection system is not required to provide summary data available on-line for decision making during course operations; this eliminates the need to make a priori judgments about summary categories. Great flexibility could thus be allowed in the off-line processing and analysis of the response data.

Specific Plan: Response Analysis

For each level of the preliminary course (see course outline earlier in this chapter), a list has been compiled of all terms, concepts, and skills the student is to learn. Course authors specify, for each response a student is to make, which of these terms, skills, or concepts is related to that response. They designate what specific characteristics of the response are to be associated with which terms, skills, or concepts. For example, a student's Data Division entry might be analyzed for errors in level number designation, picture clause, punctuation, and margins.

A student response may also be categorized in terms of:

(1) Level of confidence rating.

(2) Gross latency (from presentation of display to receipt of student response), including reading time and response time as well as

information processing time. (As noted in Chapter 2, this will be further subdivided in the next iteration as is feasible.)

(3) Student requests for information on terms.
(4) Student requests for information on compiler diagnostics.
(5) Student comments about the course, or requests for information that could not be recognized and satisfied by the present system.
(6) Level of aspiration indications.

Computer programing tools required for performing the above response analysis are discussed in Chapter 6.

Specific Plan: Data Collection

Pre-course data and student response information will be collected. Pre-course data will include biographical and pretest information (aptitude, motivation, and structure of intellect indices).

Data about the student response are recorded on magnetic tape, after the analysis of each response. The record is 404 characters long and contains the following information:

Course name.
Course segment number (in this case, level number).
Student name.
Student number.
Registration data (date student first signed on the course).
Number of days since last sign-on.
Current label-problem-sequence number (a tag which indicates the point in the course at which this response was made).
Last executed major operation code (used by Coursewriter programers for debugging purposes).
Current date.
Time student signed on today.
Label before last question.
Last question number.
Time student has spent on the course to date (in minutes).
Gross latency.
Error identification counters for this response.
Status of registers and switches (for programer use in debugging).
Switches reflecting experimental variables for this student.
Present time in minutes.
Actual text of student response.
100 supplemental characters of error identifier information (buffer 1).

Data on the student response tapes may be categorized and summarized as required by specific users—course authors, researchers, or systems programers. This involves the development of highly organized and compacted data structures that can be used by anyone who requires access to the data base. For example, examination of the above recording indicates that the information which is contained in a recording might profitably be reorganized so that specific items are grouped generically. The following list is an example of such a grouping.

Time
- Registration date (date of first sign-on).
- Number of days since last sign-on.
- Current date.

Time the student has spent on the course to date.
Present time in minutes (time of day when recording was initiated).
Gross latency.

Course Location Identifiers

Current label—problem sequence number.
Label before last question.
Last question number.
Last executed major OP code.

Student Scores

Count of unanticipated answers made by a student (per problem).
Error identification (for this response).
Actual text of student response.
Supplemental error scores.

In addition, some scores that will be useful in analyzing a student's progress through the course have been identified. These scores are not currently developed under the initial response analysis system, but can easily be obtained from the present recording mechanism. The following list is a representative sample of those items derived thus far:

(1) Total number of questions answered correctly on the first response.
(2) Total number of problems attempted in a session.
(3) Total number of standard exits from a problem sequence.
(4) Total number of non-standard exits from a problem sequence.
(5) Total number of textual pages displayed.
(6) Total number of unique problem displays.

There will be many other statistics defined and devised as more experience is gained with both course and student data analysis.

In addition, several complete statistical packages are being studied for use in the actual analysis of data. Current plans call for data analysis to be performed in batch mode, with the gradual integration of statistical routines for on-line analysis. The actual analyses of IMPACT course and student data will be described in subsequent reports.[1]

PRELIMINARY DATA COLLECTION AND EXPANSION OF CONTENT

The instruction in COBOL which has been devised by the instructional content team of Project IMPACT has been administered tutorially to a small number of trainees. The first test of the material was to be carried out in July-August 1968 on a group of students selected to represent varying abilities and whose entry characteristics have been measured.

Subject Selection for Preliminary Data Collection

Students for the summer 1968 COBOL course were selected on the basis of level of education, knowledge of computer programing, and programer aptitude. Initially, 35 subjects, at the high school graduate or college freshman level in education and naive with respect to computer programing, were recruited in the early summer of 1968. These subjects took the IBM Aptitude Test for Programmer Personnel and filled out a Personal Data Form, which provides a check on the subject's educational level and data processing background as well as routine administrative information.

---

[1] A more complete description of the on-line IDM is presented in Chapter 2. The list processor required to build the data structure above is described in Chapter 6.

On the basis of the programer aptitude test scores a group of 12 subjects was selected with test scores distributed as follows: "A" range—3 subjects; "B" range—2; "C" range—3; "D" range—4. These 12 subjects were to take part in the Entry Characteristics Testing Program.

Entry Characteristics Testing Program

Subject entry characteristics will be measured in the following four areas: (a) structure of intellect factors, (b) motivation, (c) verbal skills, and (d) computer programing aptitude.

The Entry Characteristics Test Battery is made up of 42 timed tests that can be administered in group testing sessions. The tests range from two minutes to 60 minutes in testing time, with the majority requiring less than 20 minutes. Most of the tests are of the paper-and-pencil variety but a few require special visual or auditory presentation of test items.

For the first group of 12 subjects, all entry characteristics testing was to take place off-line. The schedule called for five hours of group testing per day for three days.

At the end of the Entry Characteristics Testing Program, four students were scheduled to start the COBOL course immediately, receiving six half-days of instruction. The remaining eight, in successive groups of four, will be scheduled to take the course later.

Individualization of Instruction

The Summer 1968 testing program was planned, in part, to try out a single version of a portion of the COBOL course on students whose entry characteristics have been measured in detail. On the basis of their performance, modifications will be made to further tailor the instruction to the individual's needs.

Under this plan there has been minimal effort, so far, to individualize the course material. However, the following exceptions have been incorporated into parts of the course:

(1) Student's entering information
   a. Less than expected. He may call for definition of terms assumed to be in the basic vocabulary of the target population.
   b. More than expected. When a pretest, taken at student's option, reveals that he is informed in a basic area, he may skip certain displays.
(2) Student's feeling about his need for information
   a. Extra information is available at student's option, even though that information does not serve a functional purpose at the moment.
(3) Student's responses to instruction
   a. Variety in feedback is provided.

Chapter 6

SOFTWARE DEVELOPMENTS AND PROJECTIONS

INTRODUCTION

The Project IMPACT development plan calls for the Computer Software staff to perform several roles. They are to develop a foundation for future CAI systems (referred to for planning purposes as Generation II); to satisfy current needs, they must also act as "tool makers" for the Instructional Programing group and other CAI research personnel, so they are concurrently involved in shorter-term course development.

The longer-range plans call for work in the areas of language development and overall system design, including the design of Time Sharing systems for CAI and other users. Meeting the more immediate IMPACT goals requires working within the constraints of the CAI language currently in use on the Project, ICAIL (IMPACT CAI LANGUAGE), which is a dialect of IBM's Coursewriter. Satisfaction of current needs entails extending the capabilities of the language and supporting systems, including removing some of the constraints which Coursewriter, and hence ICAIL, imposes on instructional development and the psychological research required to accomplish the goals of IMPACT.

A major consideration in all activities is the relative amount of carry-over from one Project generation to the next. Shorter-range activities are being designed so that the future system will evolve, without a sharp break, as the Project passes from one generation to the next.

Major Software Development Efforts

For most of this presentation, a certain degree of technical sophistication is presumed on the part of the reader, especially in regard to the IBM System 360. The purpose here is to present in some detail activities undertaken in CAI software systems as part of Project IMPACT, not to present basic information on the System 360. Some technical sophistication in concepts of the computer sciences, such as list-processing techniques and the design of interactive systems, is also assumed.

The major tangible efforts of the Software group to date are:

(1) The modification of ICAIL to operate from a cathode ray tube (CRT) display rather than strictly from a typewriter terminal.

(2) The enrichment of the ICAIL author language through design and implementation of functions, using the existing function capability in Coursewriter.

(3) The definition, design, and implementation of a set of list procedures to be used in conjunction with ICAIL and with the Instructional Decision Model.

(4) Preliminary efforts toward definitions of data structures to be used in both student and course analysis studies (i.e., the Instructional Decision Model).

(5) Preliminary definition and design of a generalized system to work toward for future developments.

In planning toward future systems, several guidelines have been established:

(1) First, in examining CAI systems and languages that are most frequently publicized, and in using Coursewriter, the conclusion has been reached that the single language approach dedicated to CAI is quite likely doomed to failure because all users need and require a multiplicity of facilities.

(2) Secondly, the current "state of the art" in the computer sciences almost demands that consideration be given to a time-sharing base for systems that are intended to be operable within a two- to five-year time frame. It has been recognized that within the context of a true time-sharing computer system, CAI users must share the system facilities with others.

Both of these considerations lead to the notion that in a truly viable system, nearly all computer programs in a public library should have at least some interactive capability. That is, one should be able to build programs for problem-solving purposes which can draw upon and use all of the available system programs. This, furthermore, presumes that these programs may have to interact with each other as agents of a human user. The time-sharing base or any other operating system per se cannot accomplish this because of the large system overhead involved.

To have such a system, some concepts of software are required that extend the state of the art. The concept of "Coherent Programing" first proposed by J.C.R. Licklider (16) and currently most fully developed by Lincoln Laboratories, Massachusetts Institute of Technology (17), is thought to be most appropriate in this regard. The final section of this chapter presents a brief description of Coherent Programing.

Technical Introduction

The ICAIL system was operational in late February 1968. The system was generated to run in both the Background and Foreground partitions under DOS, on the IBM 360/40. The Foreground system is used by the Instructional Programing group in course development. The Background system is used by the Software group for experimentation.

Several modifications were made to the system to reflect the particular hardware configuration being used, specifically the number and type of teleprocessing lines. They specify the type of lines (remote), the number of lines (two), and the type of control unit (IBM 2701) which are currently in use. As additional terminal devices are added to the system, similar modifications will be made. These modifications have been made to allow the system to run as originally designed, and do not reflect modifications or additions made or intended with respect to the use of CRT's. The CAI language development being undertaken as part of Project IMPACT is also not reflected in the changes mentioned above.

A set of procedures for initiating the ICAIL system in the Foreground partition at execution time was compiled for the Instructional Programing group. These procedures include instructions for loading the disk packs and tape drives, for initiating DOS, for starting the Foreground partition, and for terminating the execution of the ICAIL program. A card deck for starting the ICAIL program in the Foreground partition was also provided.

The ICAIL System was originally designed to operate using the IBM 1050 Typewriter terminal as its only input/output device. Since the IMPACT development plan calls for the use of CRT display terminals, ICAIL must be modified to accept these terminals. While the CRT may be used more like conventional visual aids, the typewriter is a strictly serial device. The fact that the terminal may be used in this manner has other important ramifications. For example, instructional programing may be affected, especially with regard to the form and manner of presentation. Use of a CRT also affects the use of languages, and this has clear implications in the development of a total systems approach. The following section of this chapter includes more details of the researchers' approach to the general problem of making an existing language operate using foreign devices.

While the Coursewriter system on which ICAIL is based tends to be somewhat static, the language is open-ended to the extent that a user (author) may add nonresident subroutines, called functions, to the system. The purpose of a function is to allow a course author to accomplish a specific objective within his course development where this capability does not already exist. Functions for ICAIL are written in Assembler Language Coding (ALC) to specifications developed jointly by the computer programers and instructional programers. A later section of this chapter contains details on those developed thus far within Project IMPACT.

Much of the work in development of CAI languages and systems concepts has come about through using ICAIL for immediate course development. There is also an obligation to evolve a better CAI language. These aims are both explicit and implicit throughout this report. However, studies of Coursewriter (ICAIL) and other similar CAI languages suggest that what is really required in CAI is a different approach. Rather than thinking of a specific language for CAI, a functional view of the operations to be performed by both author and student needs to be taken.

Basically, an author must have a means to store text and have it retrieved for the student in some specified order.[1] A student, however, may need to retrieve certain information that is pertinent to his learning process, but not necessarily part of the text being presented in a predetermined instructional sequence. Both author and student may need to rely on computational facilities, such as matrix routines, which are not really part of the CAI package or the instructional sequence.

Toward this end, a concept of the visibility of various languages (and their processors) and other tools which are included in the system has been developed. This concept includes the notion that not all users of a system need, nor should they have access to, all of the programs available in the system. It has been found that having list-processing capabilities available allows a pedagogical freedom heretofore unattainable within the constraints of Coursewriter.

Recognition of a need for some list-processing capabilities in ICAIL is a major step in the progress of the IMPACT CAI software development. It will facilitate the recording of student response information (including background studies) in a form usable by course authors and researchers who are unskilled in developing computer system software. No claim is made that the list-processing primitives explicated in this report constitute a language. They do not, because there are no rules for linking together the operators to comprise well-formed expressions in an open system which is consistent. There is no syntax other than that used in passing parameters to a primitive call, and no semantic context. Any meaning stems solely from the user who issues the calls, and hence the use of such a processor is meaningful only in a specific context. Those acquainted with current list-processing languages will, however, see a similarity between the process just discussed and SLIP (18). Appendix E of this report includes the implementation details of the IMPACT list processor.

---

[1] Questions are really a special form of text, and need not be considered separately here.

The Software group has also undertaken some preliminary work on the IDM, particularly on data structures to be used in collecting and storing the information to be used in the model. The efficacy of using list-processing techniques for these purposes has been shown. They are particularly appropriate because a user, without being a computer programer, can use the list-processing primitives effectively. That is, the researcher who is not a programer can have easy access to the required data without too much difficulty, and without having to acquire sophisticated programing skills. A preliminary version of the work on data structures for the IDM is given later in this chapter.

ICAIL—CRT CONVERSION

The ICAIL system based on the IBM Coursewriter is designed to handle only the IBM 1050 Typewriter terminal as the author-student input/output device. A fundamental decision that typewriter terminals would not be sufficient to attain overall Project IMPACT objectives was made at the outset of the project. This decision, of course, has many ramifications for instructional programing as well as for a computer software development. Orderly development of this phase of the project called for a critical examination of the nature of both the CRT and the typewriter terminal devices, to gain a clear understanding of their fundamental characteristics and differences.

First, it should be noted that a typewriter is a strictly serial device and forces an order of course presentation that is, in effect, superimposed on the sequencing of instruction. That is, any typewriter or similar terminal forces a sequential presentation of text, no matter what presentation might be dictated by the logic of the course. Another critical difference is the fact that a typewriter line is not necessarily analogous to a full display on a CRT when data are transmitted. A single transmission either to or from a CRT terminal may contain many more characters than a single transmission using a typewriter as the terminal. That is, the number of characters transmitted is a function of the display capability of the terminal device. In addition, data to be displayed on a CRT can often take on additional structure (e.g., blocks and pages), which is not possible when a line-at-a-time device such as a typewriter is used.

The Sanders 720 Terminal which is being used by IMPACT allows the formating of text in a page-block structure, as well as a line at a time. This is partly due to the 1024-character (maximum) delay line storage that holds all characters to be displayed at any instant in time on the CRT face. The additional structuring capability is built into the hardware logic of the terminal. This capability seems to be the major distinguishing feature between the two devices. Thus the CRT can be used like a blackboard or other visual aids in a classroom instructional situation.

The first (and most naive) approach to the CRT conversion was to increase the size of the line control block (LCB) in the ICAIL programing system for each terminal. This area contains the buffers that hold the student responses. The primary student response buffer could simply have been recoded to hold 1024 characters. However, upon further examination, it was found that this area is referenced by many routines and from many different places in ICAIL.

Ordinarily this might not make any significant difference, but in the IBM System 360 assembly language it would cause potentially serious code displacement (instruction alignment) problems. It was then realized that modification of the entire data management system, including disk addressing, as well as cylinder, track, and record organization, would be necessary.

The ICAIL data management system is closely related to the author language, a situation that exists in any interactive system. These interrelationships, and those between internal modules and subroutines, were examined in detail; the intricacy of the programing system led to the conclusion that modification of ICAIL internals was not a feasible course of action.

In a final analysis, central processor and storage considerations make terminal interface in this case a problem of data management. In particular, the data management problems are those associated with variable length records. This stems from the page-block structure of the CRT. Since the interface is with ICAIL, there are serious implementation problems, because ICAIL is built around a data management system for fixed length records of only 100 characters, and a CRT transmission can be up to 1024 characters long.

Further study indicated that it was not sufficient to be concerned only with the nature of the devices and/or the ICAIL data management system. Since ICAIL is a monolithic system, it was necessary to work with some of the externals of the system, for example, the author language. This permits a system design taking into account consideration of three major functional components, CAI user language, data management, and terminal language. Viewing the system in terms of these major functional components and placing them in their proper perspective in the overall system led to the conclusion that course management should also be modified.

Introduction of the user-language factor also led to the realization that use of the CRT as a visual aid gives credence to the notion that presentation of text is logically separable from the remainder of CAI programing considerations, such as response recording. Asking questions and subsequent checking of the student response is also logically independent of textual presentation. Separation of text and course logic will be noted to have influenced a large portion of the design to implement the software development for the CRT conversion.

The notion of separating textual material from course decision-making operations was, therefore, adopted as being the most meaningful and comprehensive solution to the ICAIL-CRT conversion problem. At a superficial level, this amounts to the recognition by an intermediary translator of certain Coursewriter operation codes, namely, QU, RD, and TY.

The text is perceived by ICAIL as part of the command. It is important to remember that a question is simply a special form of text. Therefore, the capability of making the distinction between the imperative and the interrogative is needed only at course execution time.

The decision was to build a translator that would recognize these operation codes and then interface with the list processor. This means that the list processor becomes a tool to be used immediately in two areas: (a) in the general area of separating text from the remainder of the instructional program; and (b) as the agent through which ICAIL can most easily and comprehensively be converted to operate using CRT terminals rather than typewriter terminals. Data management problems are solved because linkages in a list handle variable length records. The translator in this case is simply a recognition device, used to intercept certain ICAIL operation codes and access the list processor before the ICAIL subsystems are activiated. What the process amounts to is "fooling" the system into thinking that it is receiving an acceptable string of text, when

General Overview of ICAIL —
List Processor Translator

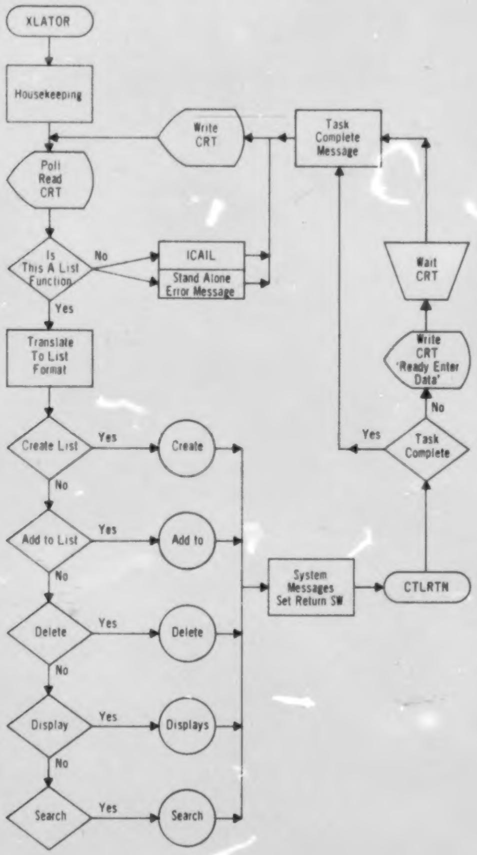

XLATOR
Housekeeping
Poll Read CRT
Is This A List Function
No
Yes
ICAIL
Stand Alone Error Message
Write CRT
Task Complete Message
Wait CRT
Write CRT 'Ready Enter Data'
No
Task Complete
Yes
Translate To List Format
Create List
Yes
Create
No
Add to List
Yes
Add to
No
Delete
Yes
Delete
No
Display
Yes
Displays
No
Search
Yes
Search
System Messages Set Return SW
CTLRTN

Figure 8

what it actually receives is a call on the list processor. Figure 8 indicates how this scheme works. The details of the IMPACT list processor are described in Appendix E.

The translator intercepts text from the terminal before it is received by the ICAIL data management system. Our concept is not, however, to intercept and then accept text per se. Since the list processor is being used as an interface, what we must receive along with any textual operation code is an author-defined call on a list primitive. That is, instead of an author typing text, his input is of the general form:

⟨operation code⟩ ⟨delimiter⟩ ⟨list primitive call⟩.

The design is such that any ICAIL operation code is amenable to the treatment afforded those which most specifically deal with text. It also means that the translator does not work simply on the basis of operation code recognition. Rather, it will activate the list processor upon any occurrence of the delimiter followed by a list primitive call. Further, it allows the list processor to function in the CAI programing system as a nearly stand-alone module, which can be operable outside of the ICAIL context.

Throughout this discussion, it must also be remembered that the original system has not been modified or completely bypassed. As shown in Figure 8, the textual string (i.e., the delimiter and the list primitive call) do get passed to ICAIL, and ultimately stored as part of the course. In this way, ICAIL gets a string, which it expects, but not one which exceeds 100 characters. The data management modules in fact remain inviolate.

The specific calls appear as follows:

Operation code [Parameter 1, Parameter 2, Parameter 3, Parameter 4, Parameter 5] where:

(1) The brackets [] indicate the string delimiter.

(2) Parameter 1 is the list name.

(3) Parameter 2 is the list position number or the entry name.

(4) Parameter 3 is the cursor position. This is a block number within which the cursor is to be positioned.

(5) Parameter 4 is an optional "no clear" condition code. If it is present, the screen will not be cleared before displaying the text associated with the operation code and list primitive call. If it is not present, the CRT face is to be cleared before display of the associated text.

(6) Parameter 5 is a "Roll-On" block number indicating the number of the block in which the text specified by the previous parameters is to be inserted.

It should be clear that what ICAIL receives is the string beginning with the first character of the operation code and ending with the right hand bracket. That is, the list procedure calls are built in the course in the same manner as the operation code is normally coded. During course execution in student mode, the operation codes are checked before they are executed to determine whether the list procedures are to be called. If so, the normal execution of the course is interrupted and the list procedures execute the command.

The following provides an example of a list of displays to be entered and displayed, where LISTNAME=LIST: ENTRYNAME=ENT

```
          CREATELIST [LIST]
          ADDTOLIST [LIST,ENT]
          DATUM IS ENTERED HERE
          DISPLAYS [LIST,ENT,3]
      LIST,ENT schematic
          #AAAAAAAVTAAAA#VTVTBBBBB#VTVTVT____#
where A = characters in Block 1            ↑
      B = characters in Block 2
      _ = blank spaces in Block 3
```

The cursor will be positioned in the first noncontrol type character position in Block 3 see ↑.

FUNCTIONS

In order to gain a fuller understanding of the use of functions in ICAIL, it should be noted that there are some fundamental differences between 1440 Coursewriter (on which ICAIL is based) and ICAIL itself. The major difference is that, in 1440 Coursewriter, functions are available as a standard part of the language, while in ICAIL, the author must design and code his own functions. In addition, ICAIL permits the use of up to 30 counters, while only 10 are provided in 1440 Coursewriter.[1]

---

[1] Some knowledge of 1440 Coursewriter and its use is assumed.

ICAIL author language gives a user the ability to extend the power and usefulness of the language to fit specific requirements. This is accomplished primarily through the FUNCTION (FN) operation code. Functions are nonresident subroutines that are designed and then coded by the user in ALC. The use of functions gives the author capabilities that are not normally available with the standard operation codes. ICAIL, unlike 1440 Coursewriter, has no functions that are a standard part of the language. The author must determine which necessary capabilities for his particular instructional environment are not available with the standard operation codes. The Software group is responsible for working out a viable design with the course author, and for all of the actual implementation details.

The author uses the FN operation code in his course just as he would any other minor operation code. The operand field of the FN operation code specifies the name of the function. Any parameters or arguments that are needed are coded in the manner prescribed by the actual implementation. To the ICAIL system, parameters or arguments are simply text.

When the course is being executed and the FN operation code is encountered, the subroutine designated by the function name is called and executed.

It will be observed from the following descriptions of functions LIST, SEEK, AND SEEKP that functions SEEK and SEEKP provide essentially the same capabilities, and that they both fully incorporate all of the capabilities provided by function LIST. A convention has been adopted whereby functions will be left in the system once they are there, even though a function that is written later may incorporate the services provided by the earlier function. This approach gives the author the additional capabilities required without having to re-code portions of the course that make use of the earlier functions.

The function LIST which has been developed for use in ICAIL was written to enable carry-over of the course in basic assembly language programing developed under the Work Unit METHOD. This course was originally developed using the 1440 Coursewriter system. It has now been converted to ICAIL and hence requires compatible software support.

It should be noted that function LIST has no relationship to the list processor discussed in the preceding section and in Appendix E. This function is one which was written to solve a specific problem encountered in coding the METHOD course for the 1440 Coursewriter system. It is usable and meaningful only within the context of the Coursewriter language and the ICAIL dialect.

Function LIST for 1440 Coursewriter is really two functions, function LIST "old" and function LIST "new." Function LIST "new" compares a student response with a list that is the textual part of a CA instruction. After the response is compared with the list, the list which is part of the CA instruction is written on a work disk. Function LIST "old" retrieves the list from disk, compares the response with the list, and then writes the list on disk again. In either case, if the student response matches one of the list items, that item is deleted from the list and the up-dated list is written on disk. If no match occurs, the list written on disk is the same as it was originally.

Each item in the list has a one-character alphanumeric identifier. When a match occurs, this identifier is placed in a student counter where it can be referenced by the author in a determination of the sequence of student responses. Counter 9,2,1 has been arbitrarily specified as the counter that will contain the identifier of the matched item.

Function LIST for ICAIL also compares a student response with a list of anticipated responses, deletes the matched item from the list, and places the identifier of the matched item in a counter. However, there is only one function

LIST for ICAIL. The list is neither the text of a CA instructor nor is it retrieved from disk. The author loads the list of anticipated responses into one of five student buffers via an LD operation code. The list is not written on disk. Counter 29 has artibrarily been chosen as the counter that will contain the identifier of the matched list item.

Functions SEEK and SEEKP are like function LIST (for ICAIL) since they compare a student response with a list of anticipated responses that has been loaded into one of the five student buffers. These functions neither retrieve lists from disk nor write lists on disk. Unlike function LIST, both SEEK and SEEKP will compare a student response consisting of more than one response item with the list of anticipated responses. These functions accumulate the number of response items that match list items. The author may specify that matched items are to be deleted from the list or he may elect to leave the matched items in the list.

Function SEEK allows the author to specify which counters he wishes to use as depositories for the identifiers of matched list items. If the author does not choose to specify these counters the function will place the identifier of the last matched item in counter 28. The author may also specify which counter he wishes to use to accumulate the number of correct response items. If he does not specify a counter, the function uses counter 29 as the accumulator.

Function SEEKP was written to give the authors the capabilities provided by function SEEK without having to use counters, since the counters are often more urgently needed for use as counters as such, rather than as depositories for list item identifiers. Thus, function SEEKP uses positions in the buffers to hold identifiers and as accumulators. The author must specify which buffers and which positions in the buffers are to be used. The function has no default characteristics.

Figure 9 shows the major similarities and differences between functions LIST (for 1440 Coursewriter), LIST (for ICAIL), SEEK, and SEEKP.

Similarities and Differences Among Four Functions

	List (1440)	List (ICAIL)	SEEK	SEEKP
Compares a one-word response with a list of responses	✓	✓	✓	✓
Compares a response of more than one word with the list			✓	✓
The list may be text of CA op code	✓			
The list may be retrieved from disk	✓			
The list is written on disk	✓			
Deletes matched items from the list	✓	✓	✓	✓
Author may choose to leave matched items in the list			✓	✓
Places identifier of matched items in a counter		✓	✓	✓
Author specifies counter for identifier			✓	
Accumulates number of correct items			✓	✓
Author specifies counter for accumulator			✓	
Author specifies buffer position for accumulator				✓
Author specifies buffer position for identifier				✓

Figure 9

A function RECORD has been developed to allow recording of student response information upon author demand, rather than have it always accomplish implicitly within the system.

INSTRUCTIONAL DECISION MODEL DATA STRUCTURES

The purpose of this section is twofold. First, it will serve to illustrate the manner in which it is envisioned the data used by the IDM will be structured. Second, it will address the operation of the model itself. While the model is clearly viewed as ultimately being implemented by means of a set of computer programs, the data structures developed can be used externally to these programs. That is, computer implementation is not a prerequisite, nor will all data in this structure comprise all active parameters of the initial IDM. The fact that there are some defined data structures for the model means that the operational implementation of the model's characterization has been seriously considered. However, there is no computer implementation yet.

The IDM has a central role in the IMPACT system, and it is to be equated to the instructional agent. However, it is not to be the only instructional agent, since it will be invoked by course authors and researchers as well as by the instructional programer, especially during these early cycles of the Project.

The aim is to provide, as well as possible, individualized instruction for each student, based on his current state of learning and taking into account individual and personal characteristics affecting the learning process. Therefore, there must be a store of information about the student that is pertinent to his learning process. For many operations of the IDM, this information will consist of quantifiable data (it is also quite likely that IDM data will be floating point numbers, but this is not a requirement). This is not a prerequisite, however, for the structuring and storage of student data. What is required is either that the information be stored in the form in which it is to be processed, or that necessary and sufficient transformation rules be supplied to the system. It will be assumed for the time being that the researchers will in fact store data in such form as to facilitate immediate processing.

There are essentially two classes of data that are to be structured, collected, and stored: student data and course data. Obviously it is necessary to have the student data to make the IDM work. Course data (e.g., sequencing information) is also required, at least in part, in order to proceed with instruction. It is also not beyond reason to consider the possibilities of joint data collection of both student and course responses to be used in course analytic studies, which may be extra-systematic to the IDM.

Most of the information to be stored for the IDM will be stored in some form of a list structure. The list structure provides the most flexibility, and can be modified either externally (conceptually) or internally (in the computer) more easily than can rigidly fixed structures such as arrays. From the researcher's point of view, lists can be created, modified, and deleted almost at will. At this time, what has been provided is some naming capability, so that related data can be put into lists which have names meaningful to a researcher (the names used here should be considered as examples only). Also provided is a capability for the user to access lists freely—the pointer mechanism is useful regardless of where the data are stored. The mechanics of these operations will not be discussed here, however.

At the present time, student identifier information to be stored in a list called Student Identification Data (SID) will be considered. Each entry in this

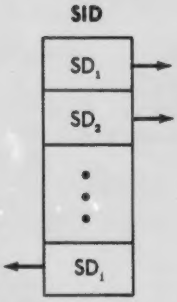


Figure A

list is really a student number, and points (relatively) to a sublist that will contain either actual data about the i-th student or pointers to the data on him. For conceptual purposes, however, we consider the SID list to contain all of the data regarding all students registered for the course. The SID list is shown in Figure A. The entries in the list are really pointers to the collected data for each student, and are called Student Data (SD). The pointers would in reality point to lists named SD_1, SD_2, and so forth, but these are shown as empty in order to indicate, simply, the construction of the main student identifier list. Operationally, this list can be derived from the ICAIL list of registered students, which provides a point of departure to set up a data collection mechanism. That is, ICAIL maintains a list of all students registered for each course, so this is used. It does not provide the list structuring mechanisms that are required for building the structures we have in mind.

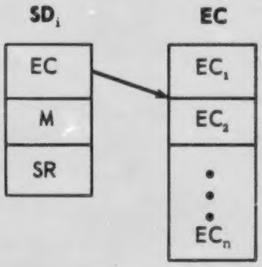


Figure B

The pointers SD_1, SD_2, . . . , and SD_i are themselves names of lists, and in fact are main lists that point to specific classes of data entries. The following entries in the list have been identified to date: Entrance Characteristics (EC), Motivational Data (M), and Student Response Information (SR). These entries are themselves lists. Figure B shows how each SD list is constructed. Even though there are currently only three entries in each SD list, the structure is flexible enough to contain as many or as few entries as might be required for each student. Figure C shows how the structure indicated in Figure B is expanded. Henceforth, however, we shall concern ourselves with only the student, so that detail will not obscure the relationships.

It has been determined that the EC list will have several components. Some have been identified, and others have not. Figure D shows how this structure will be represented, independently of any identification except for generic names.

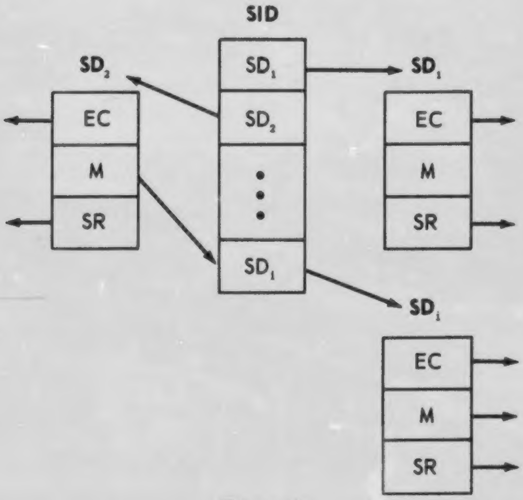


Figure C

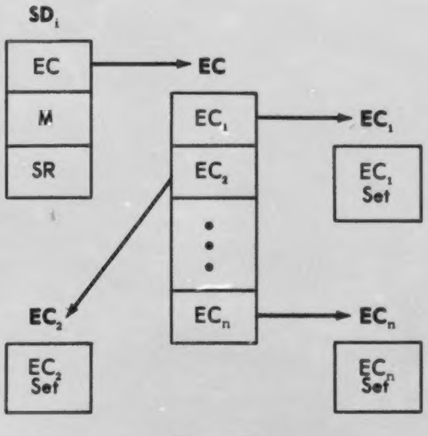


Figure D

As an example, one of the EC$_n$ components, (the first, EC$_1$) can be used. In the current structure, the EC$_1$ set becomes the Guilford Model (3, Frontispiece). This model has been conveniently represented as a three-dimensional array by Guilford; and it is useful to maintain this representation occasionally in order to decrease the number of parameter names to be considered. The data structure currently being devised will in fact allow this representation to be maintained, since the Guilford model is a sub-structure, and a means of getting to it has been developed. Figure E shows both the Guilford model as displayed in the frontispiece of the book, and the linkages by which the information contained therein is stored and retrieved.

Thus far, the discussion has been limited to only a fraction of the SID$_1$ list. Even though Entrance Characteristics are crucial, they do not constitute the whole of student identifying information. Sublists of various other (quantifiable) information must also be created. For example, there is a Motivational Data

Guilford Model or Structure of Intellect[1]

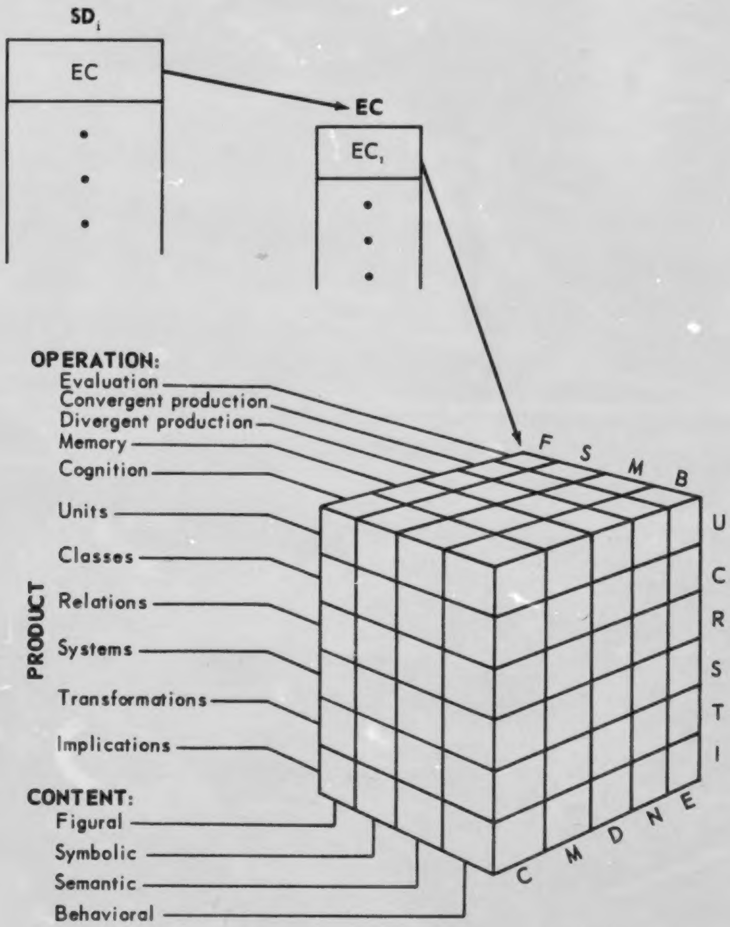

SD$_i$
EC
EC
EC$_1$
OPERATION:
Evaluation
Convergent production
Divergent production
Memory
Cognition
PRODUCT
Units
Classes
Relations
Systems
Transformations
Implications
CONTENT:
Figural
Symbolic
Semantic
Behavioral
F
S
M
B
U
C
R
S
T
I
C
M
D
N
E

Figure E

---

[1] Permission to use this copyrighted material has been granted by J. P. Guilford.

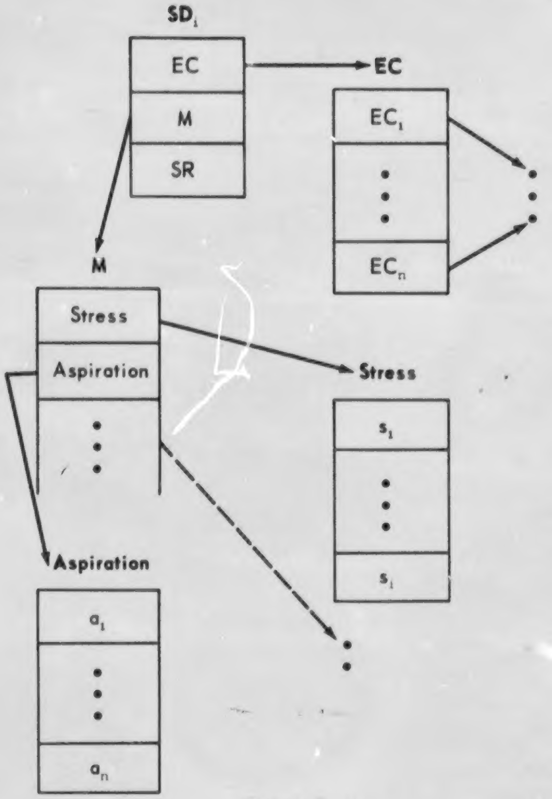


Figure F

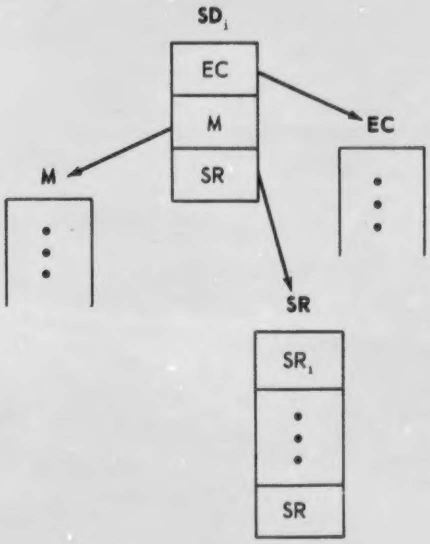


Figure G

category. For purposes of the example, we call this list M. We have included in this category closely related items such as stress levels, level of aspiration, etc. Figure F shows how the IDM Data Structure might look after this information has been added.

For purposes of exposition, the $s_1, \ldots s_i$ and $a_1, \ldots a_n$ merely represent the data collected and stored in sublists which we have called Stress and Aspiration, respectively.

The last item (currently identified) in the SD_i list is a pointer to a student response list. This set of lists should, for operational purposes, have the stature of a main list. For reference purposes, however, we must consider this set of lists as a sublist under each SID_i. The basic structure of this now more complete notion of IDM data structures is illustrated by Figure G. The name of the student response list is SR.

As examples of some of the data which can be collected under the SR list, it is considered that data currently collected under ICAIL are prime candidates for the SR list. Since there are currently 28 separate items being recorded, it is easy to see how the SR list will grow, and the items need not be enumerated here.

Figure H is an illustration of the entire data structure as we have defined it throughout this section.

It should be noted that nothing has been said about operations on this structure. From a computer software point of view, the currently perceived operations are the list operations defined in the second section of this chapter. In addition, some means of naming variables and parameters will be needed in order to make the structure itself accessible by the research staff. Decision-making logic external to ICAIL is also thought to be imperative, and is in fact implicit in the creation of abilities to deal with named variables. It can be seen

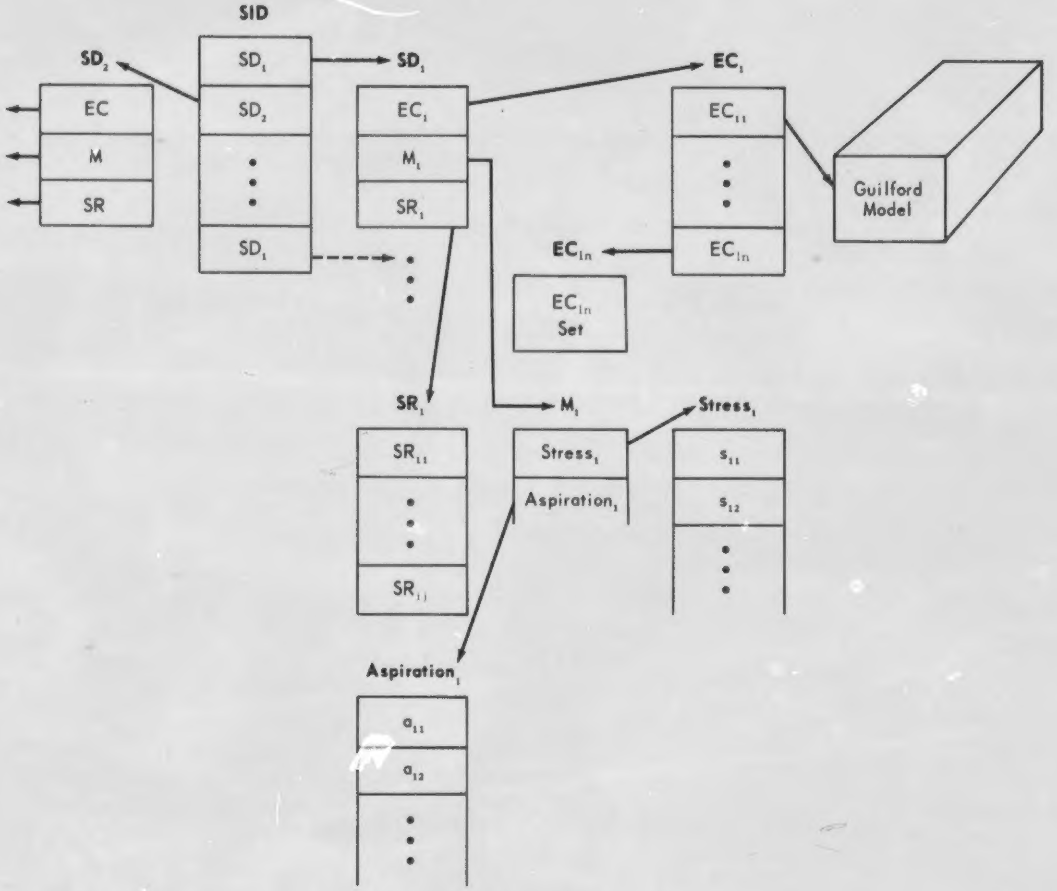

SID
SD₂
EC
M
SR
SD₁
SD₂
SD₁
SD₁
EC₁
M₁
SR₁
EC₁
EC₁₁
EC₁ₙ
EC₁ₙ
EC₁ₙ Set
Guilford Model
SR₁
SR₁₁
SR₁ⱼ
M₁
Stress₁
Aspiration₁
Stress₁
s₁₁
s₁₂
Aspiration₁
a₁₁
a₁₂

Figure H

that there will have to be other information processing modules, and routines will be developed within the context of the IDM. Chapter 2 of this report indicated the general nature of these requirements.

FUTURE SYSTEMS AND CAI SOFTWARE DEVELOPMENT PLANS

As stated earlier, the Project IMPACT development plan calls for the development of two generations of CAI systems. The first generation is based on an existing CAI language and its processors, modified to reflect changes in the hardware configuration and requirements of the research staff. It is planned to have Generation II evolve from Generation I as smoothly as possible. To do this, however, requires consideration of future directions during the initial design phase of Generation I.

One of the conclusions reached at this early stage is that CAI is an obvious candidate as a user of time-shared systems. However, the user of a CAI system, whether he is a course author or student, is likely to make use of other systems facilities, such as conventional compilers, text editors, and so forth. He should be able to use them without recourse to either a complex job control language or the more difficult alternative of having to terminate a process in order to perform a relatively trivial operation. One way to avoid these problems is the notion of "Coherent Programing" discussed below. Seeing these things during Generation I does in fact give some overall direction to the Project's software design.

Coherent Programing

Coherent Programing is basically a set of conventions and techniques designed to shape the growth of a library of programs to enable the user to draw upon them freely with minimal concern about the details of their compatibility. This is a point of logical design of programing systems which underlies much of our basic philosophy. That is, each module in the system is designed with its own primary purpose in mind; and even though the potential of the module is a primary consideration, it is not a principle in the design of the module.

Coherent Programing has two objectives: (a) to provide an environment in which it is easy to add to a library of programs without disrupting what already exists, and (b) to facilitate communication between on-line services for the non-programer. Coherent Programing is also very useful to the professional programer. When a programer uses an on-line system, he is seldom operating the program on which he is working. He uses editors, debugging packages, compilers, etc., from the system library. If these programs are coherent, the programer has greater flexibility in their use and his task is simplified. For purposes of illustration, the system we have in mind is illustrated in Figure 10.

All of the systems programs which are commonly used are in a coherent public library. The Mediator is a program which allows one to achieve coherence by building the required directories, program stacks (queues), and so forth. It is also a primary goal of such a system to allow a terminal user to work in several languages, without the time delays currently encountered in a "compile, load, and go" process in a job stream under conventional operating systems.

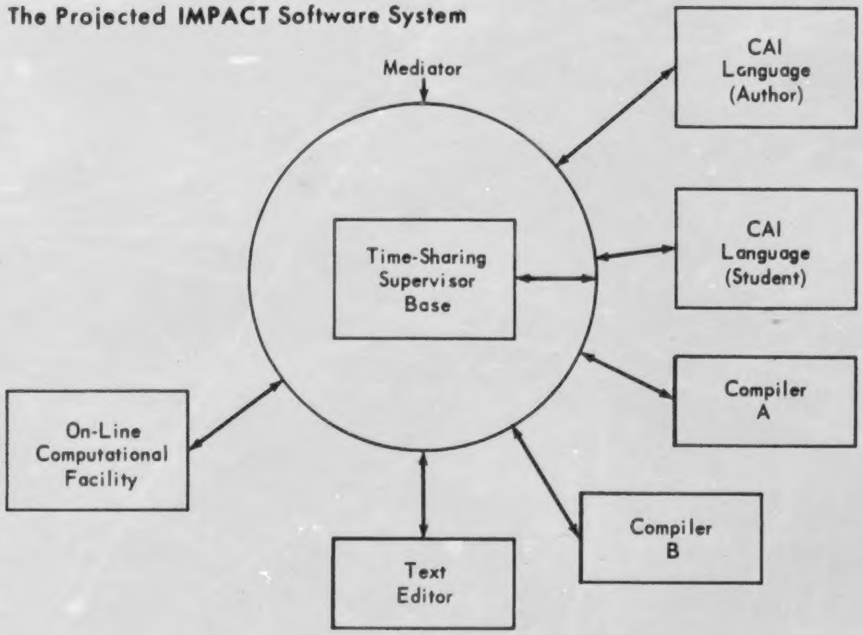

The Projected IMPACT Software System

Figure 10

The design of a Mediator system requires that particular attention be paid to the role each of the major component subsystems is to play. This is especially important in developing the required communication facilities between system modules. It is also important in those instances where programs are to act as agents for a human user, insofar as they may call on other programs or modules without human intervention.

As a simple example of this concept, consider the use of "HELP" routine which is invoked whenever a program module finds that it does not have the required input or input form. It is up to the HELP module to try to determine whether the information required by the processing module which got into trouble does in fact exist somewhere in either the public files or the human users' private files. If the information is available in the system, HELP must determine whether it can be put in proper form for the processing module, and then go ahead and attempt it, or else call another routine to perform the required transformation. In this case, both HELP and the transformation routine are acting as agents for the human user.

The notion of Coherent Programing as currently conceived, however, could itself become a subproject—a luxury which Generation I cannot afford. Therefore, the concepts are being used in a slightly modified form to accomplish current Project objectives, and at the same time provide the necessary tools and building blocks to assist in a smooth transition between Project generations.

It should be recalled that even in the design of the programs to accomplish the immediate task of the ICAIL-CRT Conversion described in the second section of this chapter, the major functional components of the current system were isolated and their role or position in the system carefully analyzed. The point of taking this approach for meeting current objectives is that it allows one to begin achieving some degree of coherence even now. It should also be recalled that, through the list processor and translator mechanisms described in the second section of this chapter, a communication link has been established between the CRT input mechanism and ICAIL. This is more than just a link, however; it is a decision mechanism and communication region that allows the author input to determine the exact calling sequence and invoke the proper system module for execution of ICAIL input statements. The fundamental difference here is that the system modules are considerably larger than those normally conceived in a time-sharing environment. In this case, ICAIL is itself a system module, as well as being a complete system.

The relationship between the future Coherent Programing system and the current CAI software development effort should be clear.

Other Factors

Factors other than Coherent Programing and time sharing also enter into Project IMPACT software development plans. Language studies are under way to help determine a direction for CAI languages and processors. The notion of taking a functional view of CAI system components has already evolved from this study, even though it is not fully completed. This has led in part to some of the current implementation design, and is also related to the concept of Coherent Programing.

Another notion that is just starting to take form is one of viewing CAI as a communication system. This involves determining the logical flow of information from an author to a student and using the Instructional Decision Model as an extension of and complement to the human author. This means, of course, that the concept of a very complex instructional agent is involved in the total communication system. We know that information flows in general from: (a) Author to student; (b) author to model; and (c) student to model and author. It can also be seen that information will flow between modules of the IDM, even though explicit details have not yet been formulated.

All of these concepts relate not only to software development as such, but also to the CAI research being done in the Instructional Decision model and in course organization and development. Future directions for Project IMPACT CAI software will take into account the needs in all aspects of CAI activities.

LITERATURE CITED

1. Seidel, R.J., and Hunter, H.G. *The Application of Theoretical Factors in Teaching Problem Solving by Programed Instruction*, HumRRO Technical Report 68-4, April 1968.
2. Seidel, Robert J., and Kopstein, Felix F. "A General Systems Approach to The Development and Maintenance of Optimal Learning Conditions," HumRRO Professional Paper 1-68, January 1968; based on presentation at American Psychological Association convention, Washington, September 1967.
3. Turing, A.M. "Computing Machinery and Intelligence," in *Computers and Thought*, E.A. Feigenbaum and J. Feldman (eds.), McGraw-Hill Book Company, Inc., New York, 1963.
4. Guilford, J.P. *The Nature of Human Intelligence*, McGraw-Hill Book Company, Inc., New York, 1967.
5. Harary, F., Norman, R.Z., and Cartwright, D. *Structural Models: An Introduction to the Theory of Directed Graphs*, Wiley Press, New York, 1965.
6. Gagné, Robert M. *The Conditions of Learning*, Holt, Rhinehart and Winston, Inc., New York, 1965.
7. Flament, C. *Applications of Graph Theory to Group Structure*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1963.
8. Berge, C. *The Theory of Graphs and Its Applications*, Wiley Press, New York, 1964.
9. Taber, J.I., Glaser, R., and Schaefer, H.H. *Learning and Programmed Instruction*, Addison-Wesley, Reading, Mass., 1965.
10. Restle, F. "A Metric and an Ordering on Sets," *Psychometrika*, vol. 24, no. 3, 1959, pp. 207-220.
11. Dunham, J.L., Guilford, J.P., and Hoepfner, R. "Multivariate Approaches to Discovering the Intellectual Components of Concept Learning," *Psychol. Rev.*, vol. 75, 1968, pp. 206-221.
12. Bunderson, C.V. *Transfer of Mental Abilities at Different Stages of Practice in the Solution of Concept Problems*, Educational Testing Service Research Bulletin RB-67-20, ONR Technical Report, Contract Nonr 1858-(15), 1967.
13. Shuford, E.H., Jr., Albert, A., and Massengill, H.E. "Admissible Probability Measurement Procedures," *Psychometrika*, vol. 31, no. 2, 1966, pp. 125-145.
14. Shoemaker, Harry A. *The Functional Context Method of Instruction*, HumRRO Professional Paper 35-67, July 1967.
15. Shoemaker, Harry A., Brown, George H., and Whittemore, Joan M. *Activities of Field Radio Repair Personnel with Implications for Training*, HumRRO Technical Report 48, May 1958.
16. Licklider, J.C.R., "Language for Specialization and Application of Prepared Procedures," in *Second Congress of the Information System Sciences*, J. Speigel and D.E. Walker (eds.), Sparton Press, Washington, 1965.
17. Wiesen, R.A., Yntema, D.B., Forgie, J.W., and Stowe, A.N., "Coherent Programming in the Lincoln Reckoner," Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, Mass., [undated].
18. Smith, Douglas K., "An Introduction to the List-Processing Language SLIP," reprinted in *Programming Systems and Languages*, S. Rosen (ed.), McGraw-Hill Book Company, Inc., 1967, pp. 393-417.

Appendix A

SUMMARY OF STAFF DEVELOPMENTS BY QUARTER

First Quarter - FY 68 (July - September 1967)

Position	On Board Prior to 1 July	Added During the Quarter
Senior Behavioral Scientist	x	
Senior Behavioral Scientist	x	
Behavioral Scientist		
Junior Behavioral Scientist (1/2 time)	x	
Junior Behavioral Scientist		
Mathematician		
Senior Software Expert		x
Software Expert		
Senior Systems Programmer		
Computer Programmer	x	
Computer Programmer		x
Computer Programmer		
Director of Instructional Programming		x
Instructional Programmer		x
Instructional Programmer		
Instructional Programmer		
Electronics Engineer		
Electronics Technician		
Computer Operator		
Research Manager		
Chief of Production & Clerical Support		
Secretary	x	
Secretary	x	

Second Quarter - FY 68 (October - December 1967)

Position	On Board Prior to 1 October	Added During the Quarter
Senior Behavioral Scientist	x	
Senior Behavioral Scientist	x	
Research Manager		x
Behavioral Scientist (3/5 time)		x
Junior Behavioral Scientist		x
Junior Behavioral Scientist		x
Senior Software Expert	x	
Computer Programmer		x
Computer Programmer	x	
Computer Programmer	x	
Director of Instructional Programming	x	
Instructional Programmer	x	
Instructional Programmer		x
Instructional Programmer		x
Secretary	x	
Secretary	x	

Third Quarter - FY 68 (January - March 1968)

Position	On Board Prior to 1 January	Added During the Quarter
Senior Behavioral Scientist	x	
Senior Behavioral Scientist	x	
Research Manager	x	
Behavioral Scientist (3/5 time)	x	
Junior Behavioral Scientist	x	
Junior Behavioral Scientist	x	
Senior Software Expert	x	
Computer Programmer	x	
Computer Programmer	x	
Computer Programmer	x	
Director of Instructional Programming	x	
Instructional Programmer	x	
Instructional Programmer	x	
Instructional Programmer	x	
Secretary	x	
Secretary	x	
Secretary		x
Secretary		x

Fourth Quarter - FY 68 (April - June 1968)

Position	On Board Prior to 1 April	Added During the Quarter
Senior Behavioral Scientist	x	
Senior Behavioral Scientist	x	
Senior Behavioral Scientist (1/2 time)		x
Research Manager	x	
Behavioral Scientist (3/5 time)	x	
Junior Behavioral Scientist	x	
Junior Behavioral Scientist	x	
Mathematician		x
Senior Software Expert	x	
Senior Systems Programmer		x
Computer Programmer	x	
Computer Programmer	x	
Computer Programmer	x	
Director of Instructional Programming	x	
Instructional Programmer	x	
Instructional Programmer	x	
Instructional Programmer	x	
Instructional Programming Technician		x
Instructional Programming Technician (1/2 time)		x
Secretary	x	
Secretary	x	
Secretary	x	
Secretary (1/2 time)		x

Appendix B

SUPERVISOR'S CHECKLIST FOR MANHOUR COMPUTATION[1]

1. Coordination and Research

a. How much customer coordination must be performed before, during, and after the program or report directive is written?

b. How much does the program or report directive relate to other projects or runs?

c. Must it be checked for compatibility?

d. Are inputs and outputs well defined?

e. Are all data elements and codes well defined?

f. Does the worker have adequate subject matter knowledge?

g. Has the worker ever done a job similar to this?

h. Are there current COBOL data divisions available?

2. Worker Skill: What level of skill best describes worker?

a. Basic trainee.

b. Advanced trainee.

c. Junior journeyman.

3. Work Environment

a. Is guidance readily available for the worker?

b. How many other projects must he work on in parallel with this one?

c. Will he be interrupted excessively while working on this project?

4. Quality of Work Desired

a. What are the consequences if the program or report directive is not correct when published or when declared operational?

b. Will any documentation be furnished to persons outside the Data Processing Department?

c. If work order pertains to a report directive, is there a requirement for a step-by-step punched card machine procedure?

d. If work order pertains to a program, is it important that the program operate efficiently?

5. Nature of Work

a. To what extent must data elements be integrated? (How many different input formats and codes are used to update how many master files or records to produce how many outputs?)

b. To what extent must logical processes be formulated?

c. To what extent must complex machine sequences be devised or traced?

d. To what extent does the project relate to new work as distinguished from revisions of previously prepared programs and report directives?

e. To what extent must exceptions be handled by machine versus manual methods?

f. Must the project be done in such a manner as to facilitate future changes and ease of maintenance?

---

[1] From Systems Branch, Fort Sam Houston Data Processing Department.

Subjective criteria most often cited by supervisors for evaluating programers are:

1. Produces programs that are easily read and easily modified. This is a matter of programing style and technique, involving at least the following:
   - Uses meaningful (to others) data names.
   - Organizes the program in separate logical units or modules.
   - Includes many comments in the source program.
   - Writes straightforward COBOL statements and routines as opposed to complex routines, or routines which take advantage of compiler idiosyncrasies. Where core storage limitations are not critical, prefers a straightforward routine which is less efficient to a complex one which would produce more efficient object code but would be harder to read and modify.

2. Adheres to installation's standards and conventions.

3. Can debug and/or modify existing programs, especially programs written by someone else.

4. In many cases, solves technical problems by using reference materials (manuals, installation memos), rather than asking for supervisor assistance.

5. Uses generalized programs and library routines where applicable.

All of the above, with the exception of the last, are incorporated in the behavioral objectives for the COBOL course. (The latter was thought to be outside the scope of the COBOL course, particularly since the generalized programs available vary greatly from one installation to another.)

Appendix C

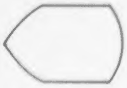
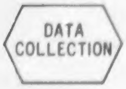
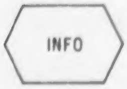
FLOW DIAGRAM FROM PRELIMINARY COBOL COURSE[1]

OUTLINE

FLOWCHART OF STUDENT-SYSTEM INTERACTION, LEVEL 2

Content Topic	Flowchart Section
Review of Level 1	A
Specifications for Problem 2	B
Flowchart Logic for Problem 2, Student Control	C - D
Flowchart Logic for Problem 2, Tutorial	E - G
Data Division, Problems 1 and 2	H - J
Identification and Environment Divisions	K - L
Procedure Division, Problem 2	M
Level 2 Test, Program Debugging	N - O

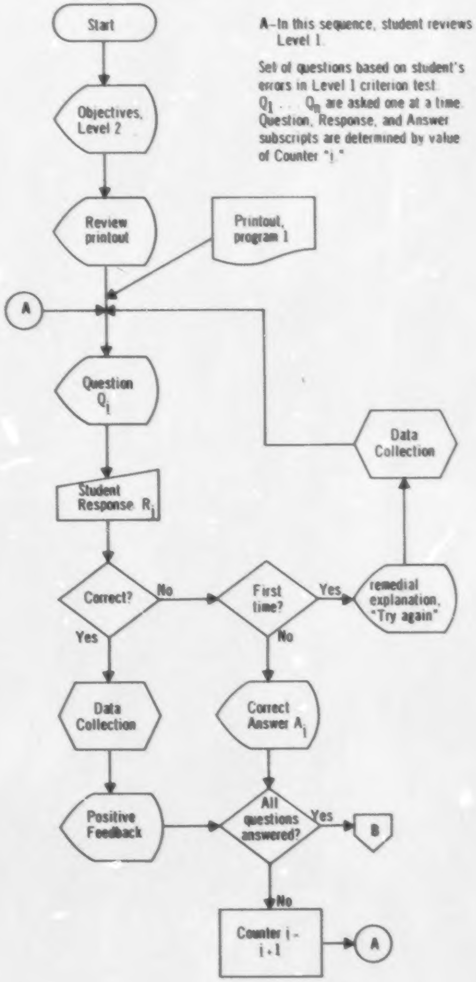
LEGEND

	A display on the cathode ray tube.
	Student input, via keyboard or light pen.
	Auxiliary display (Perceptoscope), documents, or student-generated papers, depending on context.
	Data collected on student responses.
	Student information retrieval subroutines, as described in Chapter 5.

---

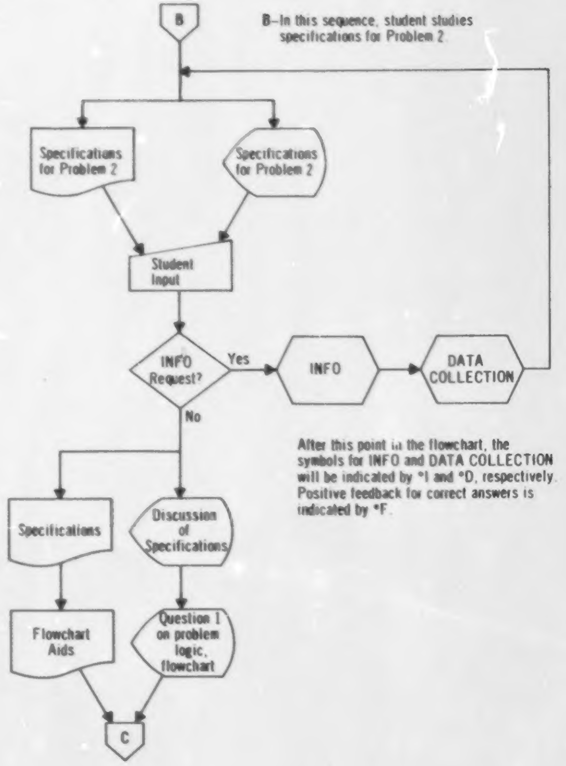
[1] At the time the report was ready for publication, the "preliminary" interactive procedures had already been revised.

FLOW DIAGRAM OF STUDENT-SYSTEM INTERACTION, LEVEL 2 — PRELIMINARY **COBOL** COURSE


Start
Objectives, Level 2
Review printout
Printout, program 1
A
Question Q_i
Data Collection
Student Response R_i
Correct?
No
Yes
First time?
Yes
No
remedial explanation, "Try again"
Data Collection
Correct Answer A_i
Positive Feedback
All questions answered?
Yes
B
No
Counter i -
i+1
A

A–In this sequence, student reviews Level 1.

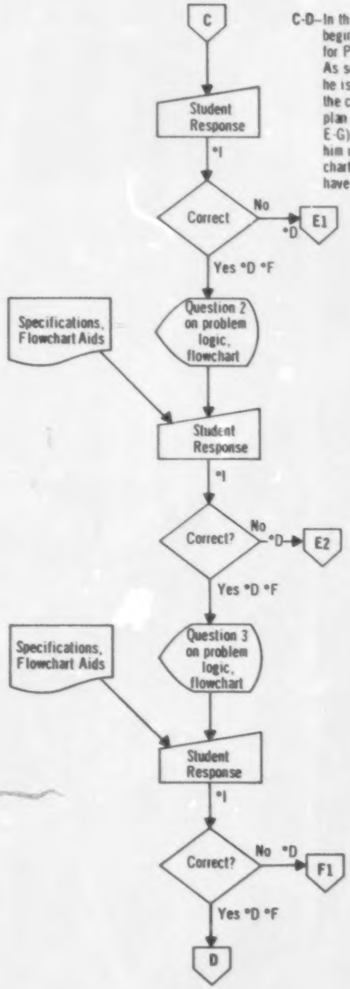
Set of questions based on student's errors in Level 1 criterion test. $Q_1 \ldots Q_n$ are asked one at a time. Question, Response, and Answer subscripts are determined by value of Counter "i."

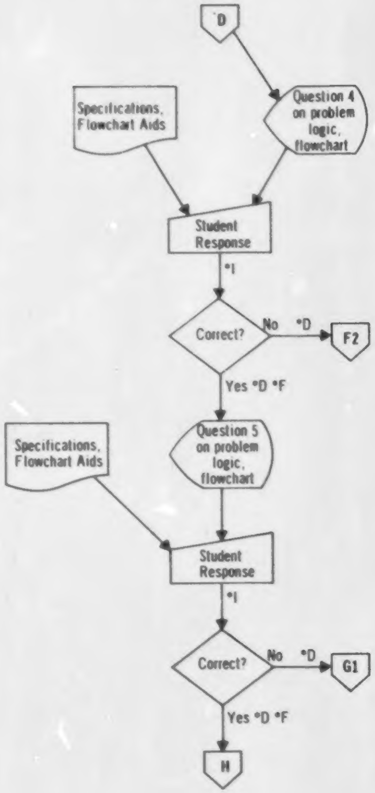

B
Specifications for Problem 2
Specifications for Problem 2
Student Input
INFO Request?
Yes
INFO
DATA COLLECTION
No
Specifications
Discussion of Specifications
Flowchart Aids
Question 1 on problem logic, flowchart
C

B–In this sequence, student studies specifications for Problem 2.

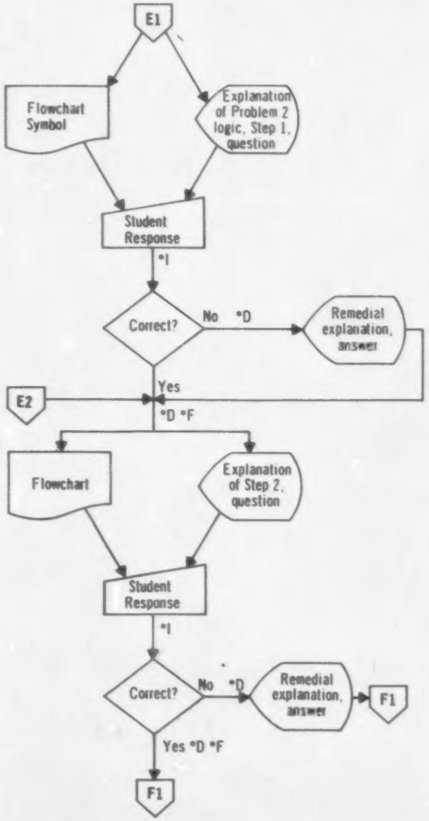
After this point in the flowchart, the symbols for INFO and DATA COLLECTION will be indicated by *I and *D, respectively. Positive feedback for correct answers is indicated by *F.

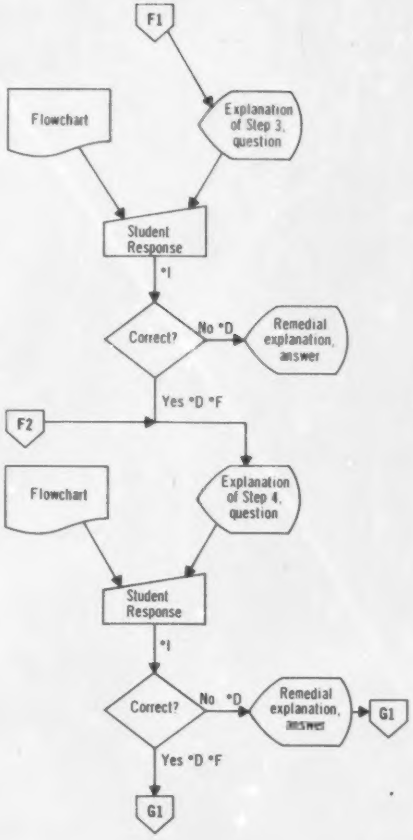
C-D—In this sequence, student begins planning the flowchart for Problem 2, without help. As soon as he makes an error, he is branched to the point in the course which helps him plan his flowchart (see Sections E-G). The branch point picks him up at the stage in his flowcharting where he began to have difficulties.

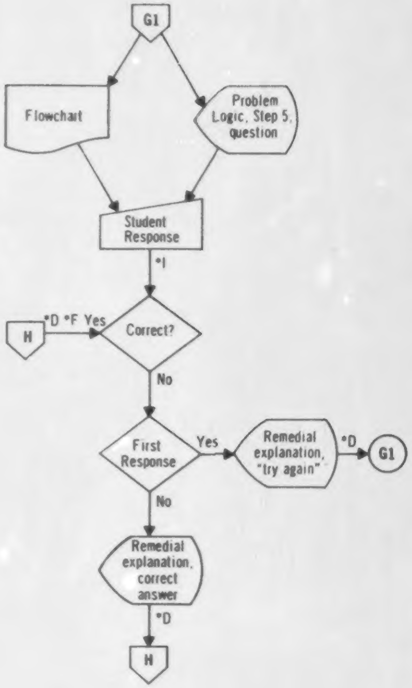

C
Student Response
*I
Correct
No
*D
E1
Yes *D *F
Question 2 on problem logic, flowchart
Specifications, Flowchart Aids
Student Response
*I
Correct?
No
*D
E2
Yes *D *F
Question 3 on problem logic, flowchart
Specifications, Flowchart Aids
Student Response
*I
Correct?
No *D
F1
Yes *D *F
D

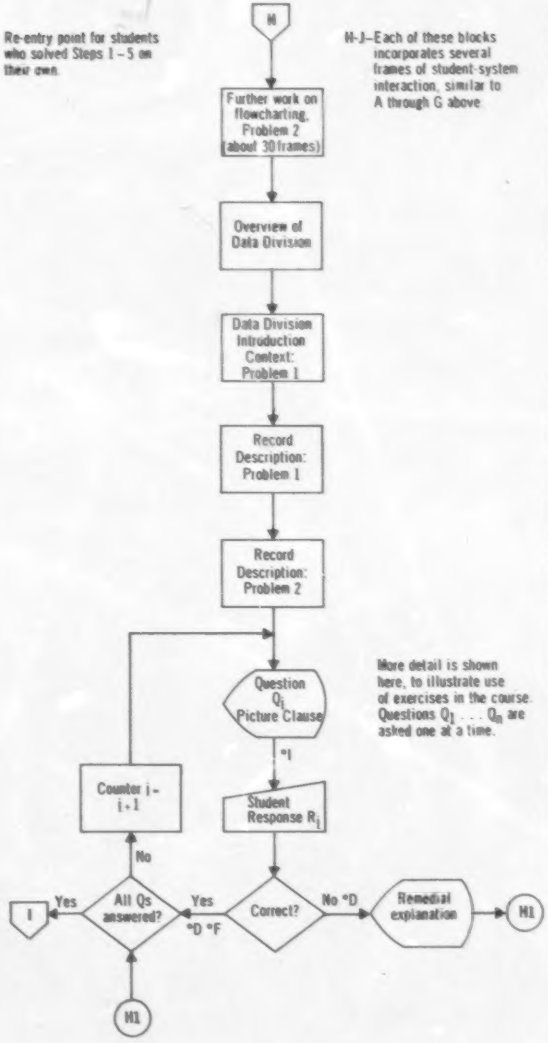

D
Specifications, Flowchart Aids
Question 4 on problem logic, flowchart
Student Response
*I
Correct?
No *D
F2
Yes *D *F
Specifications, Flowchart Aids
Question 5 on problem logic, flowchart
Student Response
*I
Correct?
No *D
G1
Yes *D *F
H

E-G—In this sequence, the student enters only when he has made an error in flowcharting the problem. If he can complete the flowcharting (C, D, and H) successfully, this remedial sequence will not apply.


E1
Flowchart Symbol
Explanation of Problem 2 logic, Step 1, question
Student Response
*I
Correct?
No *D
Remedial explanation, answer
Yes
E2
*D *F
Flowchart
Explanation of Step 2, question
Student Response
*I
Correct?
No *D
Remedial explanation, answer
F1
Yes *D *F
F1


F1
Flowchart
Explanation of Step 3, question
Student Response
*I
Correct?
No *D
Remedial explanation, answer
Yes *D *F
F2
Flowchart
Explanation of Step 4, question
Student Response
*I
Correct?
No *D
Remedial explanation, answer
G1
Yes *D *F
G1

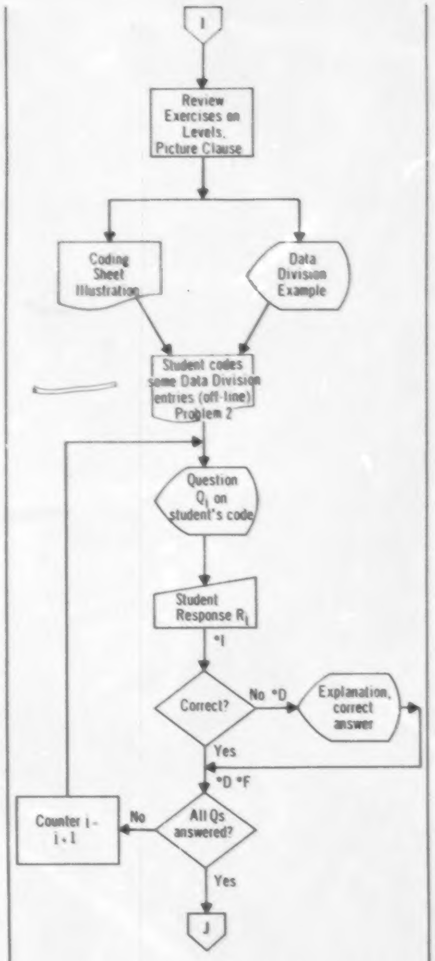

G1
Flowchart
Problem Logic, Step 5, question
Student Response
*I
H
*D *F Yes
Correct?
No
First Response
Yes
Remedial explanation, "try again"
*D
G1
No
Remedial explanation, correct answer
*D
H

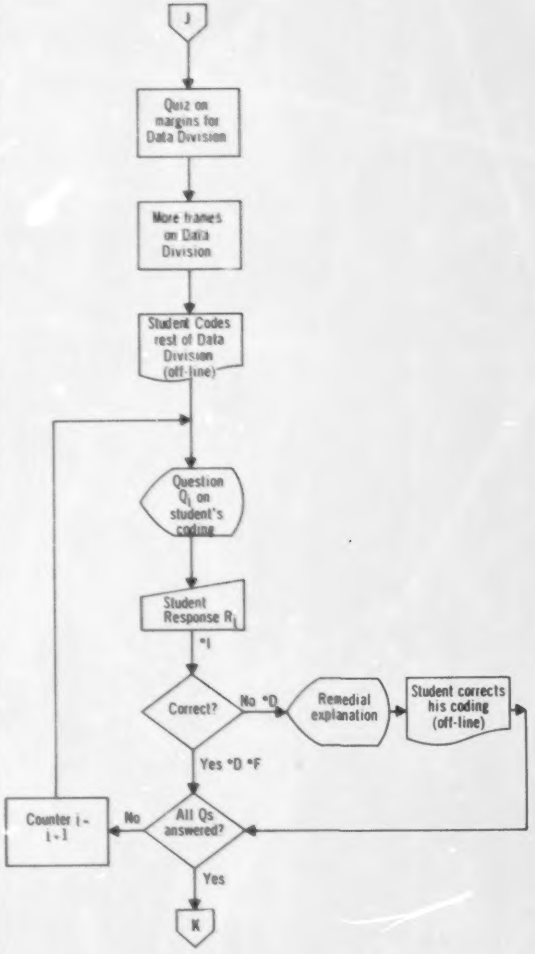

H
Further work on flowcharting, Problem 2 (about 30 frames)
Overview of Data Division
Data Division Introduction Context: Problem 1
Record Description: Problem 1
Record Description: Problem 2
Question Q_i Picture Clause
*I
Student Response R_i
Correct?
No *D
Remedial explanation
H1
Yes
*D *F
All Qs answered?
No
Counter i - i + 1
Yes
I
H1

Re-entry point for students who solved Steps 1 – 5 on their own.

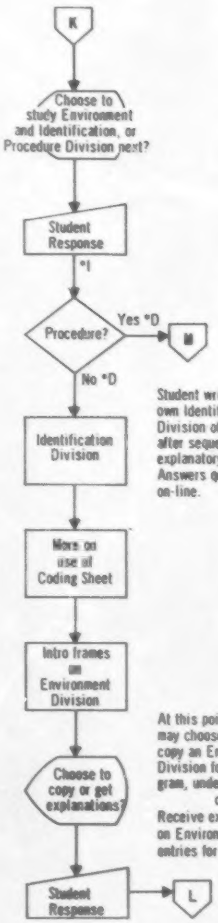
H-J—Each of these blocks incorporates several frames of student-system interaction, similar to A through G above.

More detail is shown here, to illustrate use of exercises in the course. Questions $Q_1 \ldots Q_n$ are asked one at a time.


I
Review Exercises on Levels, Picture Clause
Coding Sheet Illustration
Data Division Example
Student codes some Data Division entries (off-line) Problem 2
Question Q_i on student's code
Student Response R_i
*I
Correct?
No *D
Explanation, correct answer
Yes
*D *F
All Qs answered?
No
Counter i - i + 1
Yes
J

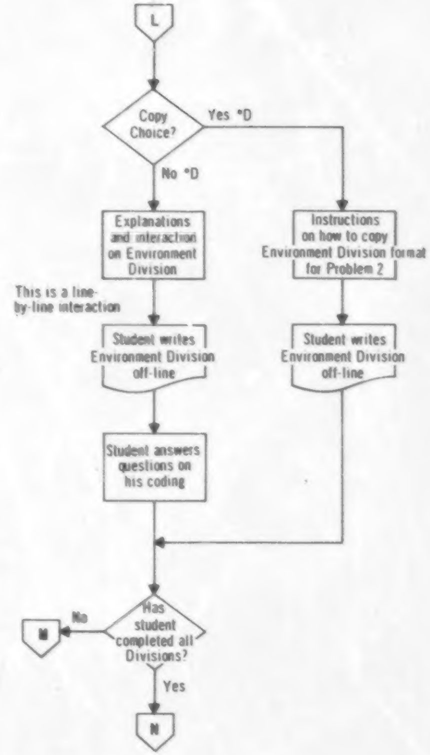

J
Quiz on margins for Data Division
More frames on Data Division
Student Codes rest of Data Division (off-line)
Question Q_i on student's coding
Student Response R_i
*I
Correct?
No *D
Remedial explanation
Student corrects his coding (off-line)
Yes *D *F
All Qs answered?
No
Counter i - i + 1
Yes
K

K-M—In this sequence, the student selects the order in which he codes the remaining divisions.

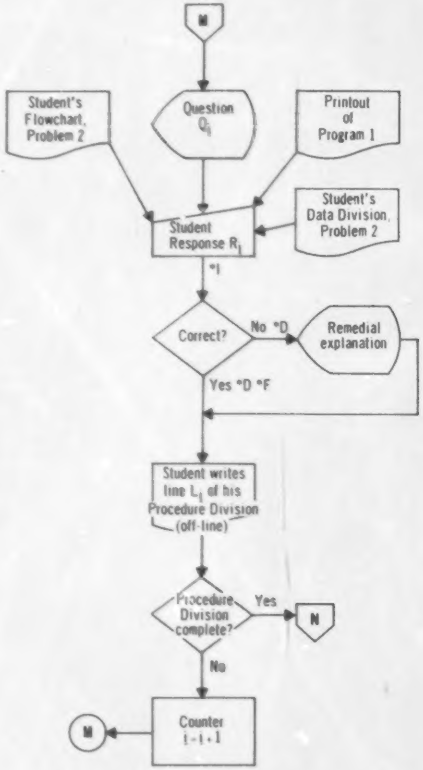

K
Choose to study Environment and Identification, or Procedure Division next?
Student Response
*I
Procedure?
Yes *D
M
No *D
Identification Division
More on use of Coding Sheet
Intro frames on Environment Division
Choose to copy or get explanations?
Student Response
L

Student writes his own Identification Division off-line, after sequence of explanatory frames. Answers questions on-line.

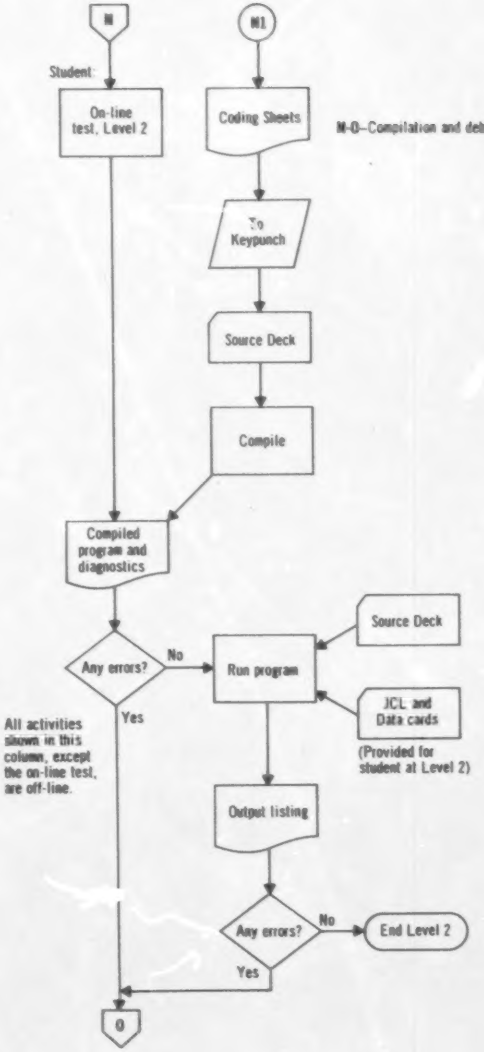
At this point, student may choose to simply copy an Environment Division for his program, under guidance,
or
Receive explanations on Environment Division entries for Problem 2.


L
Copy Choice?
Yes *D
No *D
Explanations and interaction on Environment Division
Instructions on how to copy Environment Division format for Problem 2
This is a line-by-line interaction
Student writes Environment Division off-line
Student writes Environment Division off-line
Student answers questions on his coding
Has student completed all Divisions?
No
M
Yes
N

M—Student-system interaction for Procedure Division, Level 2.


M
Student's Flowchart, Problem 2
Question Q_i
Printout of Program 1
Student's Data Division, Problem 2
Student Response R_i
*I
Correct?
No *D
Remedial explanation
Yes *D *F
Student writes line L_i of his Procedure Division (off-line)
Procedure Division complete?
Yes
N
No
Counter $i = i + 1$
M

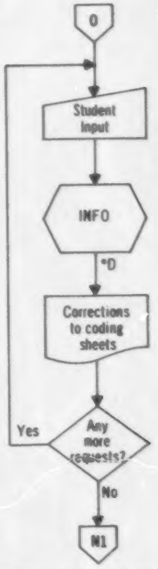
In writing his Procedure Division, the student is tutored line-by-line. When the student-system interaction of this section is complete, the student will have his entire procedure division on coding sheets.


N
Student:
On-line test, Level 2
N1
Coding Sheets
To Keypunch
Source Deck
Compile
Compiled program and diagnostics
Any errors?
No
Yes
Run program
Source Deck
JCL and Data cards
Output listing
Any errors?
No
End Level 2
Yes
O

N-O—Compilation and debugging

(Provided for student at Level 2)

All activities shown in this column, except the on-line test, are off-line.


O
Student Input
INFO
*D
Corrections to coding sheets
Any more requests?
Yes
No
N1

In this sequence, the student requests information from the system which will help him diagnose and correct errors in his program. Student may call the human proctor for assistance if he cannot locate his errors on the basis of system aids.

Appendix D

COBOL COURSE CRITERION TESTS

Table D-1

Criterion Test: Level 1

Test Item	Description	COBOL Concept Reference
A10	READ - function	READ
A20	COBOL program - number divisions	COBOL PROGRAM
A30	Record - definition	RECORD
A40	Loop - definition	LOOP
A50	PROCEDURE DIVISION - function	COBOL PROGRAM
A60	Punched cards - function	PUNCHED CARD
A70	File - definition	FILE
A80	STOP RUN - function	STOP RUN
A90	OPEN - syntax requirement	OPEN
A100	Processing one by one - necessarily also understanding COBOL coding	LOOP
A110	WRITE - function	WRITE
A120	WRITE - syntax requirement	WRITE
A130	Recognizing flowchart symbols	FLOWCHART SYMBOLS
A140	GO TO - function	GO TO
A150	OPEN - function	OPEN
A160	MOVE - function	MOVE
A170	CLOSE - function	CLOSE
A180	Output device - recognition	PRINTER
A190	Flowchart completion	BRANCHING
A200	Prompted coding: OPEN statement	OPEN
A210	Prompted coding: READ statement	READ GO TO
A220	Prompted coding: MOVE statement	MOVE
A230	Flowchart - tracing flow	FLOWCHART BRANCH LOOP
A240	Flowchart - tracing flow	FLOWCHART BRANCH LOOP

Table D-2

Criterion Test: Level 2

Test Item	Description	COBOL Concept Reference
B10	Flowchart symbol recognition	FLOWCHART SYMBOLS
B20	Picture character use	PICTURE CHARACTER
B30	Reserved words - definition	RESERVED WORD
B40	CLOSE - syntax requirement	CLOSE
B50	ENVIRONMENT DIVISION - function	COBOL PROGRAM
B60	Extract - definition	EXTRACT
B70	Flowchart symbol recognition	FLOWCHART SYMBOLS
B80	DATA DIVISION - function	COBOL PROGRAM
B90	FILLER - function	FILLER
B100	Group items - definition	GROUP ITEM
B170	Prompted coding: IF (EQUAL TO) statement	IF ---- ELSE EQUAL TO NON-NUMERIC LITERAL
B180	Prompted coding: DISPLAY UPON CONSOLE statement	DISPLAY . . . UPON CONSOLE NON-NUMERIC LITERAL
B190	Prompted coding: ADD statement	ADD NUMERIC LITERAL
B200	Data-name formation: Glossary Operative	DATA-NAME
B210	Setting up a counter: Glossary Operative	WORKING-STORAGE LEVEL 77 PICTURE CHARACTER INITIAL VALUE
B220	Writing a File description: Glossary Operative	DATA DIVISION FORMAT FILE RECORD PICTURE CHARACTER FILLER
B230	Writing a PROCEDURE DIVISION: Glossary Operative	OPEN PROCEDURE-NAME READ CLOSE STOP RUN GO TO ADD DISPLAY MARGINS PERIOD

Appendix E

THE IMPACT LIST PROCESSOR

General Introduction and Definitions

Like a set, a list is a collection of objects. These objects are related to each other only by virtue of the fact that the person who creates the list thinks about the list entries or members as being related. There is no extra-systematic logic involved, and there are therefore no prima facie logical relationships expressed other than the simple fact that the list was created as a single entity by a user. A list then, for IMPACT CAI system, is to be considered as an extension of an author's or researcher's memory within some operational constraints. It is occasionally useful if those individuals who choose to use list structures think of the lists they create as having the form of a list of things to do as a degenerate case, while keeping in mind that the forward and backward references in a list can be used to represent trees and other very complex structures or organizations. The fact that one can do some relatively sophisticated things with list structures should not obscure the fact that a list is basically an easy concept to deal with.

The purpose of the list structures which are being implemented is to allow a user to store and retrieve information relative to some base with which he is familiar, viz., the list name. All addressing is done by position in the list, that is, 1st position, 2nd position, or *i*-th position. The distinction between positions in a list and the data entry at that position is conceptually trivial, but crucial to proper use of the list primitives. Data entries are variable length character strings, with a current limitation of 3250 characters per item.

The list processing capabilities which will be included in the initial IMPACT software system do not constitute a language in the sense that LISP 1.5, IPL-V, or SLIP do, because operations cannot be concatenated to form expressions. However, one will be able to use the IMPACT list facilities to deal with symbolic data in somewhat the same manner that it is dealt with in the more commonly known languages. The elegance and translation ability offered by these languages will not be available except through some external manipulation by the user. One will, however, be able to structure and store alphanumeric information with considerable freedom.

It should be noted that theoretically a data name and a data entry can be considered as isomorphic when building sublists. That is, if LIST B is to be a sublist of LIST A, then the entry operator (an addition operator) should be able to accept the string LIST B as a data entry per se. For the initial version of these list primitives, however, where a data name is to be used it will be done so specifically. Figures A and B indicate what is meant. The instruction is ADDTOLIST [LIST A 2 LIST B]. This instruction places in the second position of LIST A a pointer to LIST B. Figure B illustrates LIST A after execution of the command.

One might easily think of simply storing the course information from the CAI system in a *n*-dimensional matrix. Indeed, some preliminary discussions on the Instructional Decision Model[1] produced the cube (3-dimensional matrix) in Figure C

---

[1] See a later section of the text chapter for a discussion of Instructional Decision Model data structures.

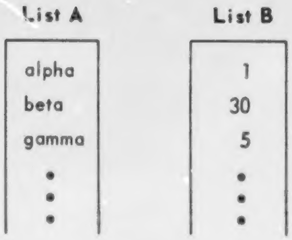

List A
List B
alpha
beta
gamma
1
30
5

Figure A

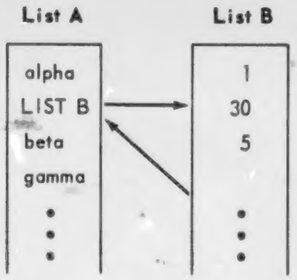

List A
List B
alpha
LIST B
beta
gamma
1
30
5

Figure B

as one representative data structure that could contain IDM data which had been defined at that date.

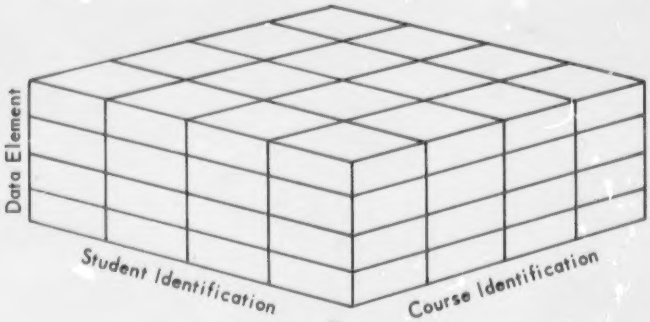

Data Element
Student Identification
Course Identification

Figure C

One dimension, the *X* axis, represents the student identification. The *Y* axis is used to represent data elements, such as response times and error rates. The *Z* axis represents course segments, identifiers, and attributes. There is nothing sacred about the *n*-dimensional matrix, except that it is a convenient means of representing data structures on a piece of paper This information can, however, be represented in the form of lists. Figure D shows how this might be done.

Student ID (*X*)	**List of Data Elements (*Y*)**	**Course Segment (*Z*)**
S0001	Response time	Phase I
⋮	Error rate	Phase II
	Correlation C_1	⋮
Snnnn	⋮	

Figure D

Primitive Operators

There are three absolute primitive operations that must be available in order to use any data structure:

(1) The ability to add information.
(2) The ability to search for information.
(3) The ability to delete information.

All of the operations currently planned are adaptations of these operations, and in this sense are here considered as primitives. It should be noted that the ability to display information either as hard or as soft copy is not a primitive operation on the data structure itself, and is therefore extra-systematic. For users of the IMPACT system, however, a set of "primitives" to allow one to display his list structure is included. A description of the primitives follows.

The first list primitive is CREATE. This takes the form:

CREATE [LISTNAME]

CREATE is the operator, and LISTNAME is an IBM legal name up to 16 characters in length. This primitive sets up a master directory entry for the named list. A CREATE command must be issued before a list structure can be operated upon.[1] As an example, consider the list of attributes of course segments. One would say:

CREATE [SEG1]

No operations are performed except creating a directory entry for the list called SEG1. Or, alternatively, some space is reserved by the system for a list to be known to it as SEG1.

In dealing with list structures, addition means adding something new to a list that has been created. Addition in this sense is therefore synonymous with insertion. That is, if one adds an item into position 2 of a list containing two or more items, the item added on the current command becomes #2, the old #2 becomes #3 and successively higher numbered items are moved down in the list. If one considers that access to data entries is gained by directory, there must be directory entries for positions 1 through *j*-1, if an item is to be added to the *j*-th position in the list. For example, if a list contains 5 items, items can be added to position 1 - 6 inclusively. An item cannot be added to position 7 since position 6 has not been filled.

There are three addition primitives designed for very specific functions. As an example, Figure E shows the status of LIST A before the execution of an ADDTOLIST A,2 command, and the list after XYZ is presented on the I/O device and after the execution of the ADDTOLIST command.

List A (before)	**List A (after)**
Position (1) ABC	Position (1) ABC
(2) DEF	(2) XYZ
(3) GHI	(3) DEF
	(4) GHI

Figure E

The forms of the ADDTOLIST primitives are:

(1) ADDTOLIST [LISTNAME, POSITION]

This primitive allows an item of data -- which will follow on the I/O device -- to be inserted in the named list in the specified position. POSITION may be a position number or a named entry.

(2) ADDTOLISTDE [LISTNAME, POSITION, DATAENTRY]

This primitive inserts the data entry -- which is a part of the command -- into the specified list in the specified position. POSITION may be a position number or a named entry.

---

[1] CREATE is also extra-systematic in the sense that adding something to a given list in the first position for the first time affects the creation of that list. We give it as a primitive for convenience.

(3) ADDTOLISTDN [LISTNAME, POSITION, DATANAME]

In this primitive DATANAME is a key word for the address of the item to be inserted or for the name of a sublist. The item or list found at the address specified by DATANAME is inserted into the specified list at the specified position. POSITION may be a position number or a named entry.

There are two DELETE primitives. The forms are:

(1) DELETEFROMLIST [LISTNAME, POSITION]

This primitive deletes the list item in the specified position from the specified list. POSITION may be a position number or a named entry.

(2) DELETEALL [LISTNAME]

This primitive deletes the specified list from the master directory. Deleting an item from a list does not physically remove the item. The list directory entry which points to a deleted item is removed. The master directory is updated to reflect the decrease in active list directory entries. Figure F shows the list before and after the command

DELETEFROMLIST [B,2]

The square brackets around the second item in List B (after) indicate that it has been deleted, as does the null operator (0) where the entry number would be.

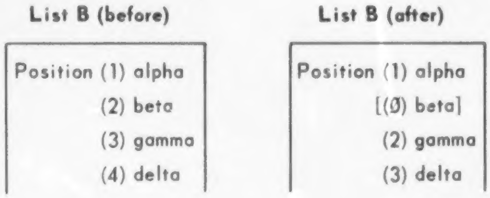

List B (before)
Position (1) alpha
(2) beta
(3) gamma
(4) delta
List B (after)
Position (1) alpha
[(0) beta]
(2) gamma
(3) delta

Figure F

Figure G shows the list directory before and after execution of the same command. The 16 byte list directory entry for beta has been removed.

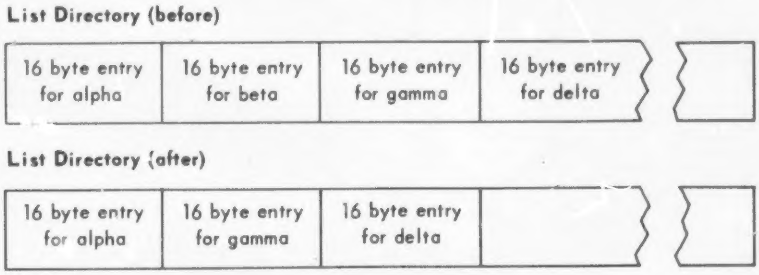

List Directory (before)
16 byte entry for alpha
16 byte entry for beta
16 byte entry for gamma
16 byte entry for delta
List Directory (after)
16 byte entry for alpha
16 byte entry for gamma
16 byte entry for delta

Figure G

Deleting an entire list does not physically remove the list from the disk. The master directory entry for the list is flagged. When the master directory is searched this entry will not be scanned. The entry and its associated list are therefore nonexistent to the list processor.

Figure H shows the master directory before and after the command

DELETEALL [B]

A description of the master and list directories is presented in the section on "Implementation Detail."

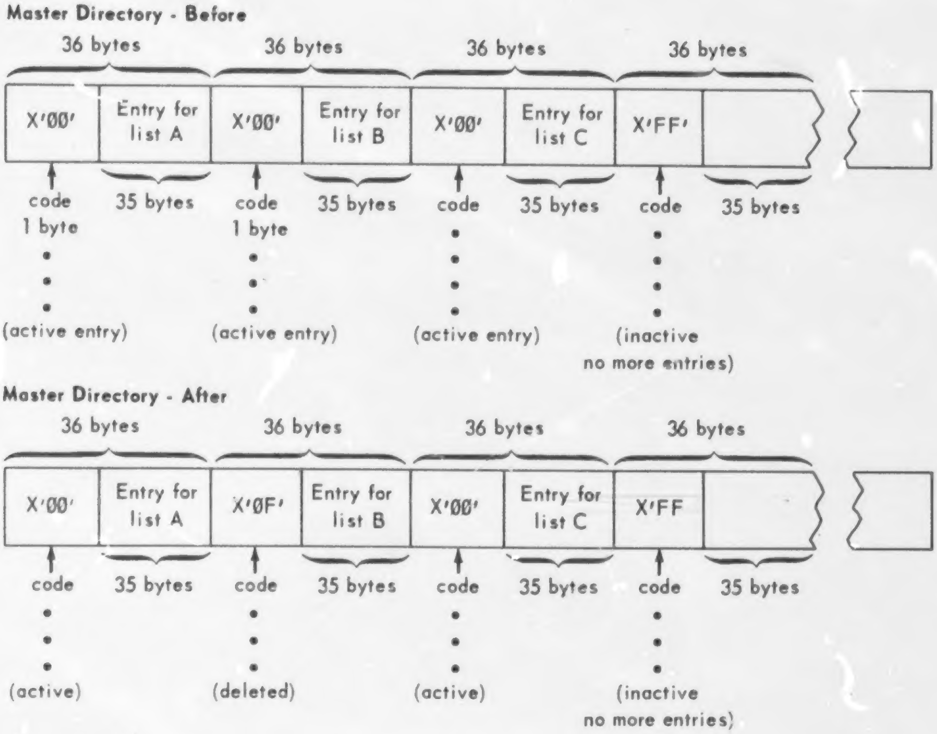

Master Directory - Before
36 bytes
36 bytes
36 bytes
36 bytes
X'00'
Entry for list A
X'00'
Entry for list B
X'00'
Entry for list C
X'FF'
code 1 byte
35 bytes
code 1 byte
35 bytes
code
35 bytes
code
35 bytes
(active entry)
(active entry)
(active entry)
(inactive no more entries)
Master Directory - After
36 bytes
36 bytes
36 bytes
36 bytes
X'00'
Entry for list A
X'0F'
Entry for list B
X'00'
Entry for list C
X'FF
code
35 bytes
code
35 bytes
code
35 bytes
code
35 bytes
(active)
(deleted)
(active)
(inactive no more entries)

Figure H

There are six primitives to display list items or entire lists. The forms are:

(1) DISPLAYS [LISTNAME, POSITION]

This primitive displays on the CRT the item in the specified position of the specified list. POSITION may be a position number or a named entry.

(2) DISPLAYH [LISTNAME, POSITION]

This primitive displays on the printer the item in the specified position of the specified list. POSITION may be a position number or a named entry.

(3) DISPLAYALLS [LISTNAME]

This primitive displays on the CRT the entire list which is specified.

(4) DISPLAYALLH [LISTNAME]

This primitive displays on the printer the entire list which is specified.

(5) DISPLAYSOFT

This primitive displays on the CRT the master directory.

(6) DISPLAYHARD

This primitive displays on the printer the master directory.

There are two primitives to search the specified list. The forms are:

(1) SEARCHP [LISTNAME, POSITION]

This primitive searches the list directory of the specified list to determine whether the specified position is in the list. POSITION may be a position number or a named entry.

(2) SEARCH [LISTNAME, DATA ENTRY]

This primitive searches the specified list to determine whether the specified data entry is in the list.

Use

It is anticipated that these list primitives will ultimately be used under the aegis of the Mediator System outlined in Chapter 6. Therefore, they are designed as stand-alone calls on the list processing routines. It is also expected that these list primitives will be interspersed with ICAIL commands. It should be noted that a list simply represents a data structure into which information is to be stored and from which the same information must be retrieved. A list processor, in its most primitive form, such as the one described above, performs no operations upon the information that is stored in this particular type of data structure. These operations are left to other processes that must occur in the system -- namely, the manipulation of information through computation, arrangement, or editing. Similarly, the movement of data from one structure to another is an issue separate from the storage and retrieval of such information.

Thus far, efforts on the list processer have been confined to the data structure and directory design indicated above. It is assumed that the user of the information will handle his own data manipulation. As examples of how the list processor may be used, however, consider a possible application within the ICAIL framework. Assume that student response information is maintained in counters and ultimately recorded in a list. Assume also that for some analytical purposes, one wished to find the mean of a subset of the counters. This process would involve the following logical operations:[1]

(1) Using a series of ADDTOLIST calls, store the contents of the counters in a list or set of lists.
(2) Search the lists for the information stored in step (1).
(3) If the information is present, accumulate a total in a counter, and in another counter record the number of occurrences of found information. Then see step (4) below. If the information is absent, ignore step (4).
(4) Compute the mean.

Note that steps (3) and (4) are totally independent of the list processor, and can be accomplished by other processing vehicles. This is a point of Coherent Programing, which should not be ignored. It is especially related to the notion of keeping the purpose or end use of a module out of the module itself. More experience in the use of list processing will come about as the Project develops more sophisticated needs and techniques. The reader is also referred to Chapter 6 for a brief discussion on the role envisioned for list structures in the Instructional Decision Model.

Implementation Detail

There will be a master list directory which contains information about the characteristics of the individual lists. Each entry in the master directory will be 36 bytes in length. Figure I is a diagram of the master directory, followed by an explanation of the fields.

Each list will contain a list directory as the first 1600 bytes. This directory consists of 100 entries, each entry being 16 bytes in length. Figure J is a diagram of the list directory followed by an explanation of the fields.

File maintenance will be accomplished through a separate program. The delete primitives will merely flag the entries to be deleted -- that is, logical deletions.

Hole-filling will not be done, so a daily condense and update of well-used files will be necessary.

Each list is a file on disk using the relocatable library of supervisor.

The procedure monitor is resident and will call the list into a predetermined area.

Each instruction will be edited before branching to the primitive function.

---

[1] Assume that the times at which these operations are performed are disjointed.

Master Directory

<table>
<tr><td>CODE</td><td colspan="4">NAME</td></tr>
<tr><td colspan="5">NAME (cont'd)</td></tr>
<tr><td colspan="3">NAME (cont'd)</td><td colspan="2">FILE</td></tr>
<tr><td>FILE (cont'd)</td><td>CYCLE</td><td>TRKE</td><td>RECE</td><td>DELR</td></tr>
<tr><td colspan="3">DISP</td><td colspan="2">RES</td></tr>
<tr><td>RES (cont'd)</td></tr>
</table>

CODE - 1 byte
A code of X'00' indicates that this is an active entry.
A code of X'0F' indicates that this entry has been deleted - the associated list is no longer available for processing.
A code of X'FF' indicates that there are no more active entries in the directory. Initially all codes are set to X'FF'.

NAME - 16 bytes
The name of the list–up to 16 alphanumeric characters.

FILE - 5 bytes
The disk address of the first position in the list–given as CCHHR (Cylinder, head, record).

CYCLE - Number of cylinders for list - 1 byte
A binary number, indicating the number of cylinders which this list encompasses–the extent of the file.

TRKE - Number of tracks being used - 1 byte
A binary number, indicating the number of tracks currently being used by this list.

RECE - Number of records - 2 bytes
A binary number telling the number of active entries in the list.

DELR - Number of records deleted - 2 bytes
A binary number telling the number of deleted list entries.

DISP - 4 bytes
This entry points to the next available position in the list - gives the number of bytes currently in the list.

RES - 4 bytes
Gives the amount of space remaining on the disk track.

Figure I

Lists will consist of physical records which occupy an entire track of the disk. When a list is brought into memory an entire track will be brought in, modified, and put back on disk.

An entry in a list may be a pointer to a sublist or to a list of lists.

General Register Conventions

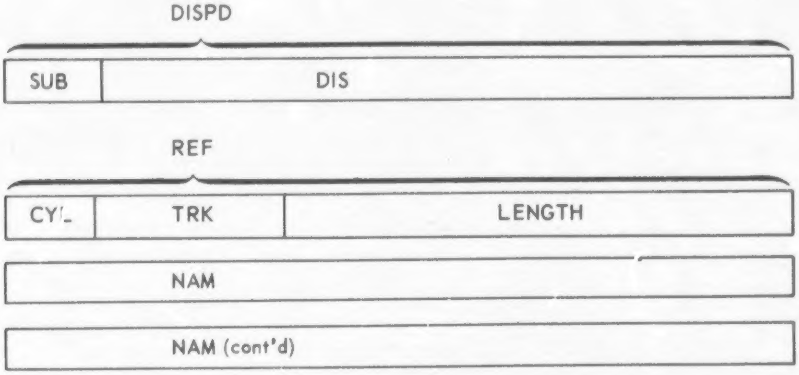
Register usage for the list procedures is as follows:

Register	Use
0	Used by DOS
1	Registers 1 and 2 will be used mainly for the TRT and TR instructions
2	

Register	Use
3	Points to LDD
4	LDA interface with Coursewriter
5	
6	Base register for master directory DSECT
7	Base register for list directory DSECT
8	Base register for parameter list DSECT
9	Disk address
10	Core address for disk get and put
11	Work register
12	Base register for subroutines
13	Pointer to save area
14	Link register
15	Branch register

Each subroutine will save and restore all registers.

List Directory



DISPD - 4 bytes–Code and displacement
    SUB - 1 byte
        A code of X'00' indicates the entry is positional.
        A code of X'0F' indicates a named entry.
        A code of X'FF' indicates a pointer to another list or sub-list.
    DIS - 3 bytes
        This entry gives the buffer displacement for the list item.
REF - Cylinder and track reference and length of entry
    CYL - 1 byte
        This entry gives the number of the disk cylinder which contains the list item.
    TRK - 1 byte
        This entry gives the number of the disk track which contains the list item.
    LENGTH - 2 bytes
        This entry gives the length of the list item.
    NAM - 8 bytes
        This entry gives the name of the list.

Figure J